CHAPTER 8

Programming ISAPI Extensions

In This Chapter

- Learning the theory behind ISAPI Extensions
- Understanding the MFC classes used for ISAPI Extensions
- Creating a simple ISAPI Extension
- Defining test suites for ISAPI Extensions that determine functionality and find bugs
- Understanding the differences between the various types of ISAPI Extensions
- Working with standalone ISAPI Extensions
- Creating an ISAPI Extension that works with scripts
- Creating an ISAPI Extension that produces user-specific content

The first thing you need to understand when reading this chapter is that the Internet Server Application Programming Interface (ISAPI) and Microsoft's Internet Information Server (IIS) are intrinsically linked. There are both advantages and disadvantages to this approach, which we'll discuss as the chapter progresses. The most important advantage is that ISAPI gives you a direct link to the functionality that IIS provides.

It's equally important to realize that you don't have to use ISAPI to get most of the functionality that IIS provides. You can work with IIS in a lot of other ways that don't involve writing DLLs. Most of them use the traditional static HTML route, which requires the use of scripts of various types. For example, you can use a common gateway interface (CGI) script to query a database through a C/C++ program. The script would then format an HTML page containing the results of the query and upload it to the client.

We aren't going to cover any of the more traditional routes of working with IIS in this chapter since there are entire books available on the topic. However, it's important to mention that the traditional methods exist because they work and you may find you need one when the newer methods we talk about in this chapter just don't fit the bill. For example, you may already have an existing infrastructure, and trying to reprogram everything using new technology may not be cost effective.

There's a fast way to learn the bare essentials of some types of IIS access techniques, including ODBC. Dynamic Systems International provides a series of lessons you can download about IIS at `http://www.dsi.org/dsi/iis.htm`. These lessons include an overview of just about everything you'll need to make some of the older technology access methods work. You may also want to visit the Microsoft IIS FAQ site (sponsored by Stephen Genusa) at `http://www.genusa.com/iis/`, which includes topics such as CGI. In addition, the KLV site at `http://www.klv.com/english/iis/` contains links to various Visual

C++ programming considerations when using older technology. The same site includes links to places where you can find out the latest information on Internet specification efforts and security concerns such as user authentication.

The Internet Server Application Programming Interface (ISAPI) represents one of the newer methods of working with IIS. However, there are other new methods to consider. For example, you can use Active Server Pages (ASP) scripts for better flexibility and to greatly enhance your ability to provide content on the Internet. We'll talk about ASP in Chapter 10, once we've learned the basics of ISAPI. However, it's important to know that ASP represents an option that many companies prefer for simple to moderately complex tasks. Since ASP relies on script, the learning curve for using it is shorter and a broader range of programmers can access it.

This chapter concentrates on how you can use ISAPI to provide great content on your Web site. Of course, one of the best features of ISAPI is that it provides greater flexibility in the presentation of information. We'll also look at specific ways to provide that increased flexibility to the users of your Web site. More importantly, we'll look at some of the enabling technologies that go with ISAPI such as the ability to combine it with scripting languages like JavaScript or VBScript. When you complete this chapter, you'll have a good overview of how this technology will work in a real-world situation. You'll also have a good idea of when you'll need to use the older technologies instead. I'll make sure to tell you about any problem areas you need to watch as you implement these new technologies on your own server. Being aware of these pitfalls may help you decide whether to go the new-technology route or stick with what you've used in the past.

Before you begin reading this chapter, you may want to spend some time learning about the client side of the picture. For example, you'll want to know how URL monikers work and what the client expects from the server in the way of direction when moving from one site to another. We looked at some of these expectations in Chapter 7, so if you haven't read that chapter yet, you may want to take a look at it. Chapter 7 will also clarify how both DHTML and XML will help you create more exciting Web pages. This chapter goes further than working with HTML/DHTML in general though. You'll want to have at least a basic understanding of JavaScript to work with the "Working with Script-dependent ISAPI Extensions" section of this chapter. This section won't provide you with any kind of JavaScript tutorial. An understanding of both ActiveX and ActiveDocument will come in handy as well, although you won't necessarily need extensive knowledge of these topics to use the information in this chapter.

The code in this chapter was developed using Visual C++ version 6.0 exclusively and hasn't been extensively tested on older platforms (the code should work fine with Visual C++ 5.0). Though the code may work without modification with Visual C++ 4.2, you'll probably need to make allowances in the procedures I've provided. The code itself will require change to work with versions of Visual C++ older than version 4.2. In most cases, you'll find it easier to upgrade your version of Visual C++ rather than try to create either ISAPI Filters or ISAPI Extensions using these older versions.

In the following sections, I'll take you through six important areas of ISAPI Extension programming. First, I'll give you an overview of ISAPI itself, which includes some information about ISAPI Filters as well as ISAPI Extensions. This is a theory section you can skip if you already have a good idea of how ISAPI works and want to get right into creating your own programs. I've purposely separated discussion of the creation of ISAPI Filters and ISAPI Extensions from their implementation in this book to make it more generic. Besides, creating an ISAPI Filter or ISAPI Extension is different from implementing it.

The second and third sections of the chapter tell you how to create a simple ISAPI Extension and then test it on a server. This ISAPI Extension won't really show you much of the power of this technology; its sole purpose is to show you the mechanics of creating any kind of ISAPI Extension. You can skip these two sections if you've worked with ISAPI Extensions in the past and want to get to the higher-end topics fast.

Don't test any of the examples in this chapter on a production system or you may experience some type of damage to your system. None of the examples in this chapter are designed for production use. They're all designed to provide an overview of how you can use ISAPI Extensions to create specific kinds of server aids. In other words, to learn how ISAPI can help you create a more flexible and higher-speed Web site, you should use these examples on a test server setup.

In the fourth section, we look at some practical examples of what you can do with an ISAPI Extension. The example shows you how to create a standalone ISAPI Extension that performs many of the same services as an ASP script, only much faster and with some added flexibility. You'd use a standalone ISAPI Extension in situations in which you have a complex task to perform and there aren't any scripting options already in place.

The fifth and sixth sections show you that scripting and ISAPI aren't necessarily mutually exclusive. The fifth section shows you how to work with an ISAPI Extension that depends on input from a server-side script. The sixth section shows you how to create an ISAPI Extension that actually creates customized script for a particular client need. Together, these sections show you that there are some ways that ISAPI and scripts can work together to create a greater whole.

One of the main reasons that you'd want to use ISAPI is that it can be more responsive to user needs than other technologies. The seventh section of this chapter will show you how to detect the current user, then create a Web page that's customized to that user's needs using an ISAPI Extension. This is an important technique to learn for company-based Web sites where each visitor has different needs and capabilities depending on the task he or she wants to perform and his or her security level. In addition, this section will help you understand the requirements for creating customizable Web sites for customer use.

There are many newsgroups you can visit to get help with your IIS, ASP, or ISAPI problem. In fact, there are too many to list them all in this chapter, so you might want to spend some additional time looking around. (It actually pays to check fairly often since new newsgroups get added every day.) For the best Microsoft-specific support for IIS, take a look at the microsoft.public.inetserver.iis newsgroup. There are other newsgroups in the microsoft.public.inetserver area that you'll want to take a look at as well, but this one usually has the most messages. You'll find Microsoft-specific ASP help at microsoft.public.inetserver.iis.activeserverpages. One of the more interesting non-Microsoft IIS sites is comp.lang.java. I was amazed to find message threads about everything from ASP to ActiveX on this site. Another good non-Microsoft site for IIS-specific help is comp.infosystems.www.servers.ms-windows. I found a great ISAPI thread on this newsgroup at the time of this writing. Needless to say, you'll want to check out other comp.infosystems.www newsgroups as well. If you're using FrontPage as one of your Web-page maintenance tools, you'll want to take a look at microsoft.public.frontpage.client for ISAPI-specific help. It's hard to find out what the bugs are in some software, but you won't find it very difficult to find out about IIS bugs if you look at comp.os.ms-windows.nt.software.compatibility. Finally, if you're searching for that hidden ASP newsgroup, take a look at microsoft.public.activex.programming.scripting.vbscript.

Understanding ISAPI Extensions

So, what is ISAPI? For the purposes of this book, ISAPI is a set of MFC extensions that allow you to work directly with IIS. We'll work with a new kind of project to implement ISAPI in this chapter, the ISAPI Extension Wizard. We'll also look at five new classes: CHttpServer, CHttpServerContext, CHttpFilter, CHttpFilterContext, and CHtmlStream.

You'll use the five classes mentioned in this section to create ISAPI Extensions and ISAPI Filters—which are called by a whole variety of other names in the Microsoft documentation, such as ISAPI Server Applications (ISAs) and ISAPI server extension DLLs. To keep things simple, we'll use ISAPI Extension or ISAPI Filter throughout the text to refer to a specific kind of ISA. However, ISA is actually a generic term that can refer to a variety of object types. All of the ISAs in this chapter rely on ISAPI. You'll always use ISAPI classes to create ISAPI Extensions and ISAPI Filters for IIS. However, you're not limited to ISAPI classes when creating an ISA; there are also WinInet classes for controlling Internet communication, and all of the standard MFC classes to provide things like an interface. In other words, it pays to be specific when talking about applications types that you can work with when using IIS.

Before you can understand ISAPI, you have to understand where it fits into the Microsoft scheme of things. There are actually five levels of Internet support provided with Visual C++, three of which can reside on the server. The other two levels of support are client-specific—you'll never see them on the server. Table 8-1 defines each level of support and tells where you can find it.

Table 8-1
Levels of Internet Support Provided in Visual C++

Technology	*Where Found*	*Description*
ISAPI	Server	This is the level of support we're talking about (at least in theory) in this chapter. You need it to provide an ISAPI Extension or ISAPI Filter for the server itself. In other words, the client won't directly interact with this level of support; it will merely see the results of the interaction. We'll discuss ISAPI Extension programming techniques in this chapter and ISAPI Filter programming techniques in Chapter 9.
WinInet	Server or Client	We don't cover this level of support in the book directly, but we do cover it indirectly. This set of classes allows you to use a specific method of data transfer between the client and the server. There are three levels of protocol support: HTTP, FTP, and Gopher. (Current versions of IIS don't actually provide a Gopher server capability.) Essentially, you'd use these classes to create a session (CInternetSession), which is one connection to the server, and then specify a connection type (CFtpConnection, CHttpConnection, or CGopherConnection). After establishing a connection, the user can do things like look for a file (CFtpFileFind or CGopherFileFind). Normally, you don't have to interact with these classes directly because Visual C++ takes care of everything for you, as we saw in Chapter 7 when creating an HTML-based client application.
URL Monikers	Server or Client	We looked at this particular area of Visual C++ Internet support in Chapter 7. The Navigate2( ) method used for the examples in the chapter relies on URL monikers. The server version of URL moniker support will be discussed in Chapter 9. The important aspect to remember is that an asynchronous URL moniker allows you to perform tasks on the Internet without waiting. You simply tell the target application what you want, and then go on doing whatever else you wanted to do. The whole idea is that the Internet doesn't provide an instantaneous response in most situations and even if it does, a long download could render the user's machine useless for hours at a time without the use of asynchronous URL monikers.

Continued

Table 8-1 *(continued)*

Technology	*Where Found*	*Description*
ActiveDocuments	Client	Displaying a document in your browser and allowing the user to edit it is what this level of support is all about. We don't visit this particular topic in detail in this book, but you'll get an overview of it in Chapter 12. For the most part, you'll find that ActiveDocument is a collaboration technology that allows more than one person to work on a document, even if they're not in the same location—they could be in a different part of the world.
ActiveX Controls	Client	Creating the basic elements of a Web page used to involve a lot of scripting, and even then you got a static image. Using ActiveX controls on a Web site means that old technology and static displays no longer hold you back—your Web page can change to meet a specific situation. The ATL form of ActiveX controls is the most popular for Web pages because this form requires less disk space and memory. We look at this technology in Chapter 12.

So far we've talked about ISAPI in very generic terms. Let's talk about the distinction between an ISAPI Filter and an ISAPI Extension. An ISAPI Filter allows you to keep something out or in by monitoring events on your server. For example, you can create an ISAPI Filter that keeps people out of your Web site unless they enter the right password. Another type of ISAPI Filter could prevent files larger than a certain size from getting uploaded to the FTP server on your Web site. ISAPI Extensions are more like applications or background processes. For example, you can create an ISAPI Extension that allows the user to interact with a database without resorting to using scripts. The same extension could create Web pages dynamically based on the user input and the contents of the database on your server. These are simple examples of what an ISAPI Filter or ISAPI Extension can do; you'd normally use them to handle complex tasks that can't be performed using simpler technologies like scripts.

You can visit a lot of places on the Internet to find out more about IIS, ISAPI, and other Internet server-related technologies such as Active Server Pages (ASP). The main Web site to visit for IIS information is `http://www.microsoft.com/ntserver/Basics/WebServices/default.asp`. This Microsoft-supported Web site contains links to just about everything you'll need to use IIS itself and even a few of the more common links for enabling technologies like ISAPI. If you

value non-Microsoft assistance, take a look at the ASP Developer's Site at `http://www.genusa.com/asp/`. This Web site contains a lot of valuable information about using Active Server Pages on your Internet server. This same site has non-Microsoft links to both IIS and ISAPI sites.

Working with the Five ISAPI Classes

Previously, I mentioned that you would be working with five new MFC classes in this chapter. Obviously, you'll continue to work with the classes you're already familiar with; these new classes provide special capabilities you need to design an ISAPI Filter or ISAPI Extension. The following list is an overview of the classes that you'll work with in the sections about ISAPI Extensions that follow. We'll work with the ISAPI Filter classes in Chapter 9. Obviously, you'll get a much better understanding of these classes when we begin to work with the example code.

CHttpServer

This is the main class that you need to use to create either an ISAPI Filter or an ISAPI Extension. The CHttpServer class defines an object that works as a server extension DLL or an ISA. The main purpose of the CHttpServer object is to handle the messages generated by the server and the client. As a result, you'll find only one CHttpServer object in any ISAPI DLL.

CHttpServerContext

The CHttpServer class actually creates this object. It's a single instance of a single request from the client. In other words, you'll see one CHttpServerContext object for every active client request. The object gets destroyed when the client request has been fulfilled. In other words, this class helps you keep track of which request you're servicing in an ISAPI Extension, even if there are multiple requests to the same ISAPI Extension at once.

CHttpFilter

ISAPI Filters require the use of a special CHttpFilter object to monitor server events. This object will look for all of the SF messages that the server generates as the client interacts with it. For example, the client will generate a SF_NOTIFY_LOG message every time a user tries to access the Web site. This message tells the ISAPI Filter that the server needs to make a log entry to register the user's access.

CHttpFilterContext

The CHttpFilter object will create one CHttpFilterContext object for each active server event. The CHttpFilterContext class represents a single event generated by a single client during a specific session. A single client request will generate more than one event, which allows ISAPI Filters to monitor the progress of a client

request through several stages of processing. In other words, this class helps you keep track of both the event and the request you're servicing in an ISAPI Filter, even if there are multiple requests to the same ISAPI Filter at once.

CHtmlStream

You'll use this class to manage data transferred between the client and the server. The CHttpServer class will create an object of this type whenever it needs to transmit information between the client and the server. In most cases, there is only one of these objects per DLL. However, the CHttpServer object can create as many CHtmlStream objects since it needs to transfer data safely. A typical CHtmlStream object contains data and all of the tags required to create the content for an HTML page. For example, if you performed a search of a database, the resulting CHtmlStream object would contain not only the data, but also the HTML tags required to actually create the Web page used to display the data. We don't really cover the use of HTML, DHTML, or XML tags in this book, but there are examples of them in use alongside the Visual C++ example code.

Using ISAPI in the Real World

There a few things you need to know about ISAPI when it comes to a real-world production environment. For example, your ISAPI Extension or ISAPI Filter will be a DLL that loads on the server, just like any other DLL. (ISAPI Filters get loaded as part of the server startup sequence due to the way you add them to the server environment; ISAPI Extensions are loaded the first time a client calls on them to perform some type of service.) The ISAPI DLL will share the same address space as the Web server, and you can unload it later if you need the memory it's consuming for something else. So what does that buy you? The following sections tell you about the advantages of using ISAPI Filters and ISAPI Extensions over other techniques such as CGI.

Lower Memory Costs

Since your ISAPI Filter or ISAPI Extension loads as a DLL on the server and uses the same memory space as the Web server, you won't have to waste the memory normally associated with CGI overhead. A CGI script loads into a separate memory space that has to be allocated from the memory available to the server as a whole. In addition, the CGI script will call on one or more C applications, each of which will require its own separate memory allocation. With ISAPI, all you'll really need to load is the logic required to perform the task you're asking the ISAPI Filter or ISAPI Extension to do. In many cases, it's unlikely that the server will require any additional memory to get the job done. Even if memory is required, the allocation requirements will be much smaller than a CGI equivalent.

Speed

Loading an ISAPI Extension DLL or C application the first time will take essentially the same amount of time, although the ISAPI Extension DLL will be slightly faster due to its smaller size. (ISAPI Filters are loaded when the server is initialized or when the ISAPI Filter is added to the server environment, which means that they require no load time when a service is requested.) However, once you've loaded the ISAPI Extension DLL the first time, it will stay in memory until you unload it, which means that you don't have to pay that loading cost more than one time if you don't want to. A CGI script will load the C application every time you call it. Even better, because the ISAPI DLL (either filter or extension) shares the same memory space as your HTTP server, you won't have to pay a time penalty for interprocess calls or any of the overhead normally associated with using a C application in a separate address space.

Reliability

C applications used by CGI scripts load and execute on the server without having much access to the server itself. As a result, it's harder to create a C application that can monitor server events and recover from errors. What usually happens is that the server will terminate an errant CGI script and the client will end up with nothing. ISAPI Filters and ISAPI Extensions have full access to the server, which means that they can recover from errors more easily. As a result, the client very seldom (if ever) ends up having to make a request the second time.

Filtering Capability

You can't provide an event-driven equivalent to an ISAPI Filter using CGI and a C application. The reason is simple: A C application gets called, does its work, and then unloads. There isn't any way that it can monitor server events over the long term. An ISAPI Filter can respond to request events during several of the processing phases, which means you have much finer control over the server's reaction to a particular request.

Multiple Tasks in One DLL

Every task that you want CGI to perform requires a separate executable. As a result, you incur the overhead of calling each routine. ISAPI DLLs, on the other hand, can contain multiple commands, each of which is a member of the CHttpServer class.

Understanding ISAPI Extension Specifics

At this point, you may be wondering what you have to pay to work with ISAPI Extensions on a Web page. The answer is nothing. Using the four capabilities described in the previous section doesn't mean that you have to pay a big price in either HTML learning curve or excess HTML coding. ISAPI Extensions are just as

easy to use as the CGI equivalent. Here's what the two lines of code that perform the same task would look like in a Web page:

```
<!-This is a call to a CGI routine with one parameter.->
http://nt_server/controls/sample.exe?Param

<!-This is a call to an ISA routine with one parameter.->
http://nt_server/controls/sample.dll?Param
```

As you can see, working with an ISAPI Extension doesn't have to be difficult. We called our ISAPI Extension DLL using about the same code as we would use for a CGI routine. In fact, from a coding perspective, the only difference is that our ISAPI Extension uses a DLL file extension, whereas the CGI routine uses an EXE file extension. Theoretically, you can switch your server to ISAPI Extensions, make a few search-and-replace changes to your Web pages, and no one would notice the difference from an interface perspective. Of course, everyone would notice the higher efficiency of your Web site due to the advantages of using ISAPI.

The last advantage that I mentioned for using ISAPI Extension was that you can perform more than one task with a single ISAPI DLL. We'll see later how you can implement this. For right now, all you really need to know is that the calling syntax still doesn't differ much from the standard CGI calls you may have used in the past. Following is an ISAPI Extension routine call in HTML that specifies that you want to use the DisplayStr function:

```
<!-Call the DisplayStr function in an ISA routine with one
parameter.->
http://nt_server/controls/sample.dll?DisplayStr?Param
```

As you can see, we called something other than the default task using a second question mark (?) in the calling string. The first parameter now states which function you want to call within SAMPLE.DLL, and the second parameter contains a list of parameters. This method of calling functions is known as a parse map, which you'll learn how to create in the "Writing a Simple ISAPI Extension" section of this chapter.

ISAPI Extensions do share some qualities that you'll find in CGI. For one thing, your application executes on the server, not the client. This makes updating your application easy — all you need to do is replace one DLL file on the server. (You do need to stop the service to update the DLL when using older versions of IIS, but this is a small price to pay for the convenience of one machine update.) Obviously, this is a lot easier than trying to replace an application on every client machine that accesses your Web site. It's also one of the reasons that companies are taking a serious look at intranets to host things like a help desk and custom database applications — updating one server is a much easier than updating many clients.

IIS version 4 adds some new capabilities that make working with ISAPI Extensions and ISAPI Filters easier. For one thing, you can tell the server to unload the DLL after each call. This means that you can try the DLL to see if it works and replace it with a new copy if necessary, all without stopping the service. The disadvantage of this new capability is that you'll see a slight performance hit because the server will need to reload the DLL every time it gets called by a client.

Writing a Simple ISAPI Extension

OK, it's time to look at our first ISAPI Extension. In this case, we'll create a very simple ISAPI Extension using the ISAPI Extension Wizard. The whole purpose of this particular example is to get you used to the idea of working with ISAPI and to show you some of the techniques you'll need to make ISAPI work. Most of the techniques you'll learn in this example are true for any ISAPI Extension you'll create — however, all production-level ISAPI Extensions will be more complex than the one shown here.

We'll look at three essential programming tasks in this example. First, we'll create a DLL that contains two functions. The first, Default( ), will display a message that explains how to use the ISAPI Extension. I normally do this with the Default( ) function so that programmers who follow won't have a hard time figuring out what I've done. The second function, DisplayStr( ), will demonstrate two tasks: working with variable information and creating an output document. The DisplayStr( ) function will accept a string from the client, change it a little, and then display a new page showing the string that the client sent.

CD-ROM Location: Chapt08\DispStr
Program Name: DispStr.DLL
Source Code Modules in Text: DispStr.CPP, DispStr.H

Now that you have some understanding of what we're going to do, it's important to know how we're going to do it. Many programmers look at creating an ISAPI Extension as an error-prone and difficult task. However, you can break the process of creating an ISAPI Extension down into easy tasks. The reality of creating an ISAPI Extension is that it doesn't need to be any more complex than creating a simple Windows application — the main difference is that you'll draw on the user's browser window rather than on a local window.

The four main programming tasks required to create an ISAPI Extension are:

1. Create the ISAPI Extension shell.
2. Define a parse map to direct the client request to a specific function.
3. Declare one or more ISAPI Extension client-request processing functions.

4. Write the code required to implement the client request processing functions, including some function-use documentation in the ISAPI Extension's Default( ) function.

Creating the DispStr Program Shell

The following procedure will take you through all of the steps required to create the sample ISAPI Extension. You can use the same procedure to start any ISAPI Extension that you want to create. I'm assuming that you already have Visual C++ version 6.0 running.

Try This

1. Use the File | New... command to display the New dialog shown in Figure 8-1. Notice that I've already selected the ISAPI Extension Wizard.

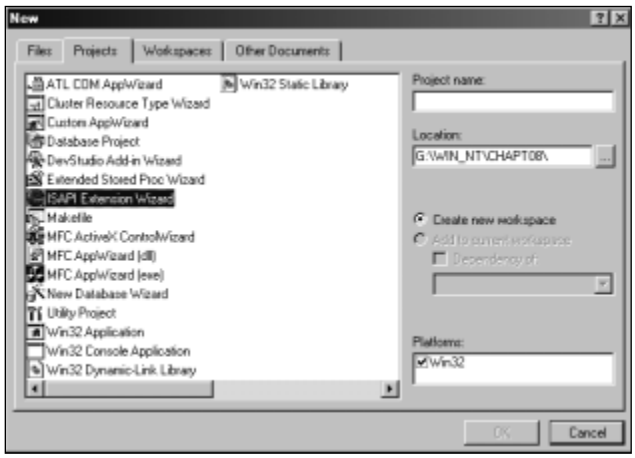


Figure 8-1: The ISAPI Extension Wizard entry in the New dialog box is used for both ISAPI Extensions and ISAPI Filters

2. Type the name of the ISAPI Extension you want to create. I used DispStr for the example ISAPI Extension, but you can use any name.
3. Click OK. You'll see the ISAPI Extension Wizard - Step 1 of 1 dialog shown in Figure 8-2. This is where you'll select the various characteristics of your ISAPI Extension. Notice that there are only three main areas of the dialog box. You can choose to create an ISAPI Filter by checking the first checkbox and an ISAPI Extension by checking the second checkbox. Microsoft sets the ISAPI Extension up as the default option. The third area defines how you'll link MFC into your application.

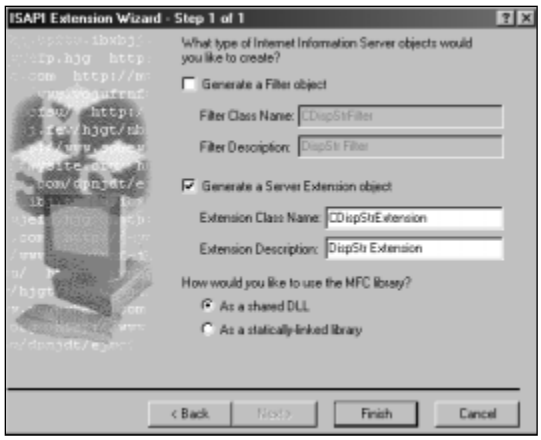

ISAPI Extension Wizard - Step 1 of 1
What type of Internet Information Server objects would you like to create?
Generate a Filter object
Filter Class Name: CDispStrFilter
Filter Description: DispStr Filter
Generate a Server Extension object
Extension Class Name: CDispStrExtension
Extension Description: DispStr Extension
How would you like to use the MFC library?
As a shared DLL
As a statically-linked library
< Back
Next >
Finish
Cancel

Figure 8-2: The ISAPI Extension Wizard allows you to chose between creating an ISAPI Filter, ISAPI Extension, or both.

Note

In Chapter 9, you'll find that we use the same ISAPI Extension Wizard shown in this section of the chapter to create ISAPI Filters. However, the settings on the initial page of the ISAPI Extension Wizard are different from the ones discussed in this section of the chapter. In addition, there's more than one type of ISAPI Filter that you can create based on the client-request events that you want to interpret. Note in Figure 8-2 that you can create a single DLL that contains both an ISAPI Extension and an ISAPI Filter, which is a good idea if the two are meant to operate together. We'll cover all of these implementation differences between ISAPI Filters and ISAPI Extensions in Chapter 9.

4. Check the "Generate a Server Extension object" option, but not the "Generate a Filter object" option (as shown in Figure 8-2). We also need to make the description a little more "descriptive." Right now it doesn't say much. You'll want to provide a short, concise statement of what your ISAPI Extension does. The description appears as a string that you can use within the DLL as needed. This description won't show up in the Properties dialog box when someone opens it for your ISAPI Extension DLL, so you'll want to add some additional text to the version information for your DLL as well.
5. Type **Display a string from the client.** in the Extension Description field.
6. Click Finish. You'll see a New Project Information dialog box like the one shown in Figure 8-3.
7. Click OK. The ISAPI Extension Wizard will create the required program shell for you.

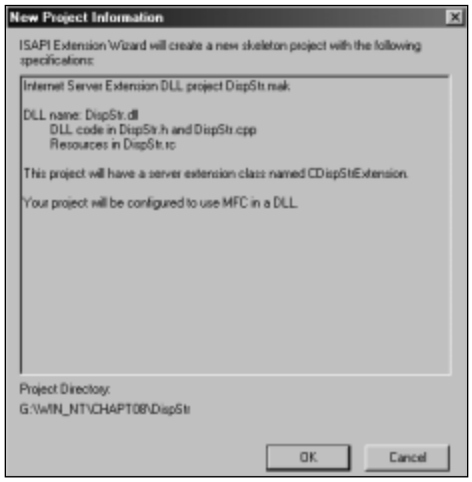

New Project Information
ISAPI Extension Wizard will create a new skeleton project with the following specifications:
Internet Server Extension DLL project DispStr.mak
DLL name: DispStr.dll
DLL code in DispStr.h and DispStr.cpp
Resources in DispStr.rc
This project will have a server extension class named CDispStrExtension.
Your project will be configured to use MFC in a DLL.
Project Directory:
G:\WIN_NT\CHAPT08\DispStr
OK
Cancel

Figure 8-3: The New Project Information dialog box displays the settings that you've chosen for the ISAPI Extension.

Defining the DispStr Parse Map

The second programming task that we talked about at the beginning of this section is to create a parse map for the new function we want to add. You'll find the parse map near the beginning of the DispStr.CPP file. Listing 8-1 shows, in bold, the code you'll need to add to the parse map so that we'll be able to access the new function from the HTML page.

Listing 8-1: **Parse Map Source Code**

```
////////////////////////////////////////////////////////////////////////
// command-parsing map

BEGIN_PARSE_MAP(CDispStrExtension, CHttpServer)
    // Our special DisplayStr parse command.
    ON_PARSE_COMMAND(DisplayStr, CDispStrExtension, ITS_PSTR)
    ON_PARSE_COMMAND_PARAMS("string='No String Supplied'")

    // Default parse command
    ON_PARSE_COMMAND(Default, CDispStrExtension, ITS_EMPTY)
    DEFAULT_PARSE_COMMAND(Default, CDispStrExtension)
END_PARSE_MAP(CDispStrExtension)
```

Even though you only had to add two lines of code to make this example work, we're actually concerned with three lines of code. The ON_PARSE_COMMAND( )

macro allows you to define a new function. Notice that we supply a function name, the class in which the function is supplied, and the type of parameters the function will use. The ON_PARSE_COMMAND( ) macro requires a parameter entry, even if you don't need any parameters to make the function work. Notice that the Default function uses a value of ITS_EMPTY since it doesn't need any parameters, but that our new function, DisplayStr, has a parameter of ITS_PSTR because it requires a string pointer.

That brings us to the ON_PARSE_COMMAND_PARAMS( ) macro on the next line. You have to tell Visual C++ how to deal with the parameters for your function. For example, if we had wanted to force the user to supply a string value for our function, we would simply say "string" in the ON_PARSE_COMMAND_PARAMS( ) macro. Because we don't absolutely have to have the user supply a string to use our function, I've supplied a default value of "No String Supplied". Be aware that the query will fail if you require a parameter and the user doesn't supply it.

Finally, you need to tell Visual C++ which function to use as a default using the DEFAULT_PARSE_COMMAND( ) macro. Since the Default function is just fine, in this case, I didn't change it.

Declaring the DisplayStr( ) Function

The third programming task you'll need to do is add the function entry to the DispStr.H file. Unless you modify the class specification to include your new function, Visual C++ won't know anything about it and the DLL won't compile. Fortunately, all we need is the single-line entry shown in bold in Listing 8-2. I've included the surrounding code so that you know where to place the new entry.

Listing 8-2: **Adding a ISAPI Extension Method Declaration**

```
{
public:
    CDispStrExtension();
    ~CDispStrExtension();

// Overrides
    // ClassWizard generated virtual function overrides
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //{{AFX_VIRTUAL(CDispStrExtension)
    public:
    virtual BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
    //}}AFX_VIRTUAL
```

Continued

Listing 8-2: *(continued)*

```
    // TODO: Add handlers for your commands here.
    // For example:

    void Default(CHttpServerContext* pCtxt);
    void DisplayStr(CHttpServerContext* pCtxt, LPTSTR pszString);

    DECLARE_PARSE_MAP()

    //{{AFX_MSG(CDispStrExtension)
    //}}AFX_MSG
};
```

As you can see, adding the function call declaration is a pretty simple matter. At this point, though, you may be wondering where the CHttpServerContext* pCtxt part of the declaration came in. We certainly didn't declare it previously in any of the parse-map macros. It turns out that the pCtxt parameter gets passed to your function by default. Remember from our previous discussion that the CHttpServer class automatically creates a CHttpServerContext object for every user request. This is where the parameter comes from. What you're getting is a pointer to the CHttpServerContext object associated with the user's call to your function. It's also the way that you keep multiple calls to your function separate — each call has a completely different object associated with it.

Writing the DisplayStr( ) Function Code

We need to do one last thing to make this DLL function — add the function code to the DispStr.CPP file. I added the function code right after the existing Default( ) function code. Listing 8-3 shows the very short function that I created for this example. There isn't anything fancy here; the purpose is to show you how to put things together.

Listing 8-3: **DisplayStr( ) Function Source Code**

```
void CDispStrExtension::DisplayStr(CHttpServerContext* pCtxt, LPTSTR pszString)
{
    // Start sending information to the new Web page
    // including a default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);

    // Display the body of the new page.
    *pCtxt << _T("<H3><CENTER>ISAPI Server Extension
Example</CENTER></H3><P><P>");
    *pCtxt << _T("This was the string you entered:<P><EM>");
```

```
    *pCtxt << _T(pszString);
    *pCtxt << _T("</EM>");

    // End the display area.
    EndContent(pCtxt);
}
```

The function itself is pretty easy to figure out. The first thing we do is tell Visual C++ to start a Web page. That's like adding the <HTML> and <HEAD> tags to a document. The second thing we do is output a title — just as if we were typing the <TITLE> tag into a document. The only way to override the default title is to override the WriteTitle( ) function — something that you can do if you'd like. Now that we've got a heading, it's time to create some body content. We have to use the stream operator to send the information. Notice that I freely use all of the HTML tags that you'll find on any Web page. Anything you can do with a standard HTML document, you can do with your ISAPI extension. We'll see in just a bit how all of these tags work together to produce a Web page. You'll also want to note that we sent the string we got from the Web page back to the new Web page — you don't even have to convert the values to text to make them work. The final function call we use, EndContent( ), tells Visual C++ that we're done sending information, which is like adding the </HTML> to the end of the document.

This example is designed to show the very basics of working with ISAPI Extensions. In the examples that follow, I'll show you some "fit and finish" programming techniques you can use to make your ISAPI Extension output look better. We'll cover things like how to create the WriteTitle( ) function and how to format your output so that it looks better and is easier to debug.

As previously mentioned, I normally use the Default( ) function to document the other functions within the ISAPI Extension. That way, if someone else tries to use your ISAPI Extension without calling one of the functions inside, he or she will get the usage instructions required to actually make the DLL work. Listing 8-4 shows the modified version of the Default( ) function that I created for this example.

Listing 8-4: **Default( ) Function Source Code**

```
void CDispStrExtension::Default(CHttpServerContext* pCtxt)
{
    // Start sending information to the new Web page
    // including a default title.
    StartContent(pCtxt);
    WriteTitle(pCtxt);
```

Continued

Listing 8-4: *(continued)*

```
    // Display the body of the documentation page.
    *pCtxt << _T("<H1>DispStr DLL Usage Instructions</H1>");
    *pCtxt << _T("This DLL contains only one function. You call");
    *pCtxt << _T(" it using the following syntax:<P>");
    *pCtxt << _T("<EM>DispStr.dll?DisplayStr?[Input String]</EM>");

    // End the display area.
    EndContent(pCtxt);
}
```

At this point, you can compile the ISAPI extension we've created. Once you successfully compile it, move the DLL to your Web server. There are several logical places to put the DLL, but the two most common are a Scripts directory or a special Controls directory. I normally keep all of my controls in one place in a Controls directory to make them easy to locate, but the actual location you use isn't all that important. The only criteria is that the user be able to access the directory containing the DLL through your Web site, and that you've marked the directory as executable using the Internet service manager provided with your Web server.

One of the best places to look for ideas for your own ISAPI Extension or ISAPI Filter is freeware produced by other programmers. The alt.comp.freeware newsgroup lists quite a few of these offerings. In addition, you'll want to check on many of the other Web sites that have appeared in the chapter so far. For example, at the time of this writing, AAIT Incorporated introduced a new freeware product named CGI Expert. It supports the CGI, win-CGI, ISAPI, and NSAPI interfaces simultaneously. Obviously, you'll want to use freeware products, like any other product, with care. However, they do provide excellent ideas on how to create your own custom extensions when needed (or they provide a solution so you don't have to do any programming at all).

Testing the ISAPI Extension

Now that you have a new ISAPI Extension loaded on your Web server, it's time to test it out. Remember that our ISAPI Extension has two functions: the default function provided by the ISAPI Extension Wizard (Default) and the one that we added (DisplayStr). In addition, our DisplayStr( ) function will react one of two ways depending on whether the user provides data for the function to work with. Listing 8-5 shows the HTML code you'll need to create a test Web page for our ISA. I used the name Extend.HTM for the resulting file.

CD-ROM Location: Chapt08\Extend
Program Name: Extend.HTM
Source Code Modules in Text: Extend.HTM

Listing 8-5: **ISAPI Extension Test Web-page Source Code**

```
<HTML>
<HEAD>
<TITLE>ISAPI Extension Example</TITLE>
</HEAD>
<!-Create a form to display our pushbuttons.->
<FORM NAME="MyForm">
<!-Display a heading.->
<H3><CENTER>Display a String ISAPI Extension</CENTER></H3>

<!-Display an edit box.->
Type a string value:
<INPUT TYPE=TEXTBOX NAME="StringValue" SIZE=40><P>

<!-Display a couple of buttons.->
<CENTER>

<!-Perform the default action.->
<INPUT LANGUAGE="JavaScript"
    TYPE=BUTTON
    VALUE="Default"
    ONCLICK="window.location.href = 'controls/DispStr.dll?'">

<!-Perform a special action with a string.->
<INPUT LANGUAGE="JavaScript"
    TYPE=BUTTON
    VALUE="Display String"
    ONCLICK="window.location.href = 'controls/DispStr.dll?DisplayStr?'
        + MyForm.StringValue.value">

</CENTER>
</FORM>
</HTML>
```

Though Internet Explorer is perfectly happy displaying <INPUT> tags that appear in the body of an HTLM document (between the <BODY> and </BODY> tags), Netscape Navigator won't even recognize them. To make a form usable with all browsers, you must place the <INPUT> tags within a form (between a <FORM> and </FORM> tag pair).

The sample code isn't that complex. It consists of a heading, a textbox, and two pushbuttons. The textbox will contain the string value that the user wants to display. Notice that we don't include a submit action for our form, but use standard pushbuttons and JavaScript to initiate an action instead. This arrangement seems to work better than a submit when using an ISA that contains more than one function, as ours does.

There are a couple of things you should note about the JavaScript provided for the ONCLICK event of each pushbutton. Notice that with the Default pushbutton we only call the DispStr.DLL itself—we don't specify a particular function. On the other hand, the Display String pushbutton includes both a function name and a string parameter as part of the DispStr.DLL call. We'll see how this works shortly.

HTML doesn't send any text after a space in a textbox to the server. For example, if you type Hello World in the textbox, the server will only receive "Hello." You must place plus signs (+) between words to send them to the server. For example, if you send Hello+World to the server, it'll see "Hello World" (the Web server log will still show the + sign, which indicates the + sign was replaced with a space after it arrives at the server). This means adding some text manipulation code not shown in the example program to replace the spaces in the textbox with plus signs before sending the string to the server.

Open the test Web page and you'll see something like the page shown in Figure 8-4. As previously mentioned, this is a test page so, in order to keep things simple, I didn't add too much formatting code.

Make absolutely certain that the directory you use to store DispStr.DLL on the server is marked for execute privileges from within the Internet services manager for your Web server. Otherwise, you'll see any number of errors from the browser as it attempts to execute the file—most of which won't tell you that it can't execute the file (the most common error is that the service isn't supported). Marking the directory for execute privileges within the operating system isn't sufficient to allow a visitor to your Web site to use the ISA you've created. In most cases, you can avoid any problems by ensuring that the ISA DLL appears in the script directory for your Web server (assuming it has one).

Let's test the DispStr.DLL. Begin by clicking the Default button. You should see a default Web page that we created to document the DispStr ISAPI Extension like the one shown in Figure 8-5.

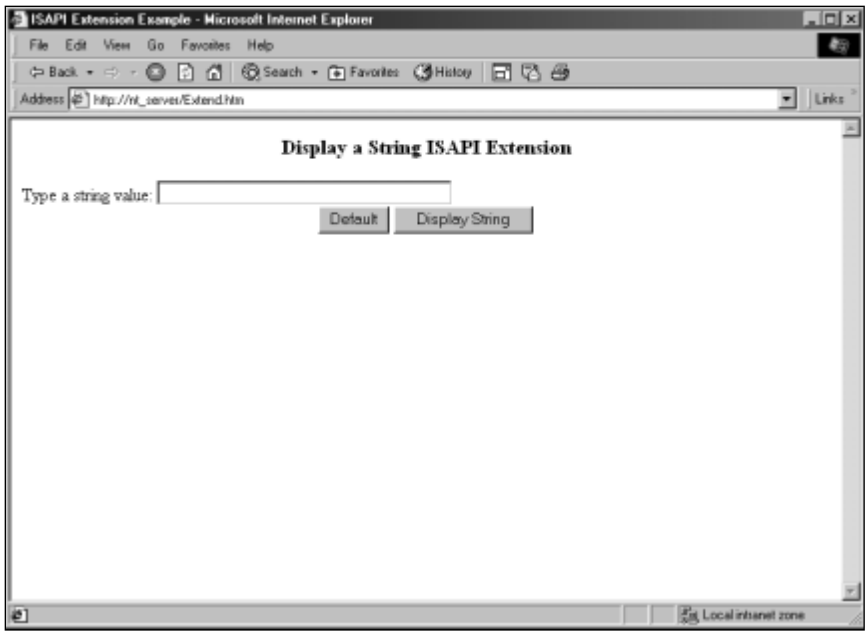


Figure 8-4: This simple test page allows us to check the full capability of DispStr.DLL.

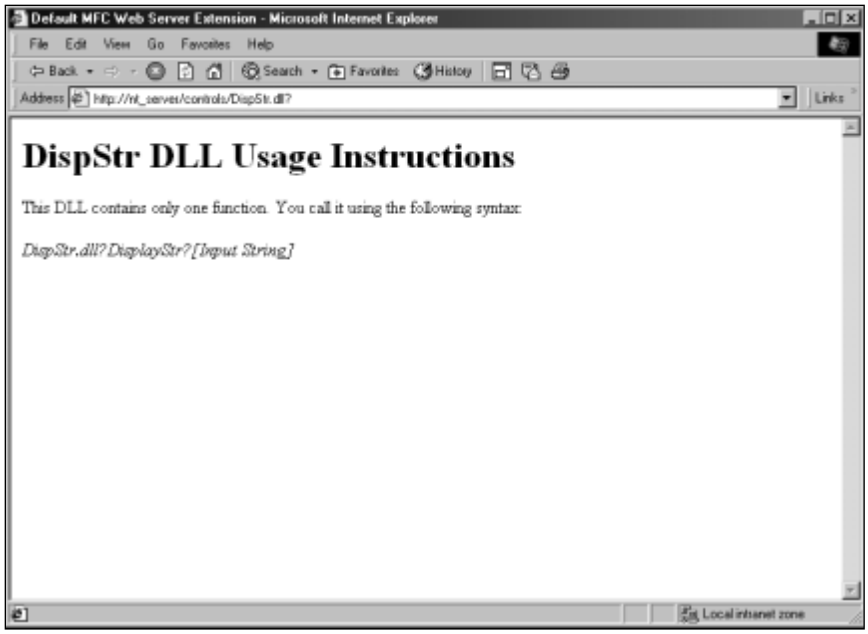

Default MFC Web Server Extension - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/controls/DispStr.dll?
Links
DispStr DLL Usage Instructions
This DLL contains only one function. You call it using the following syntax:
DispStr.dll?DisplayStr?[Input String]
Local intranet zone

Figure 8-5: Calling DispStr.DLL without any parameters automatically displays the documentation Web page.

Click the Back button (or the equivalent on your browser) to get back to the Extend.HTM page. Now click the Display String button and you'll see something like what appears in Figure 8-6. In this case, we called DispStr.DLL with the DisplayStr function, but since there wasn't any value in the textbox, the function had to display the default string we defined as part of the ON_PARSE_COMMAND_PARAMS( ) macro. Now you can see the importance of providing a default value in most cases. Users don't have to worry about your ISA causing problems if they forget to provide a needed value. What they'll get instead is a return value that could help them troubleshoot their Web page. In such a case, the user will know precisely what went wrong and can fix the problem in a matter of minutes instead of the hours that debugging normally takes.

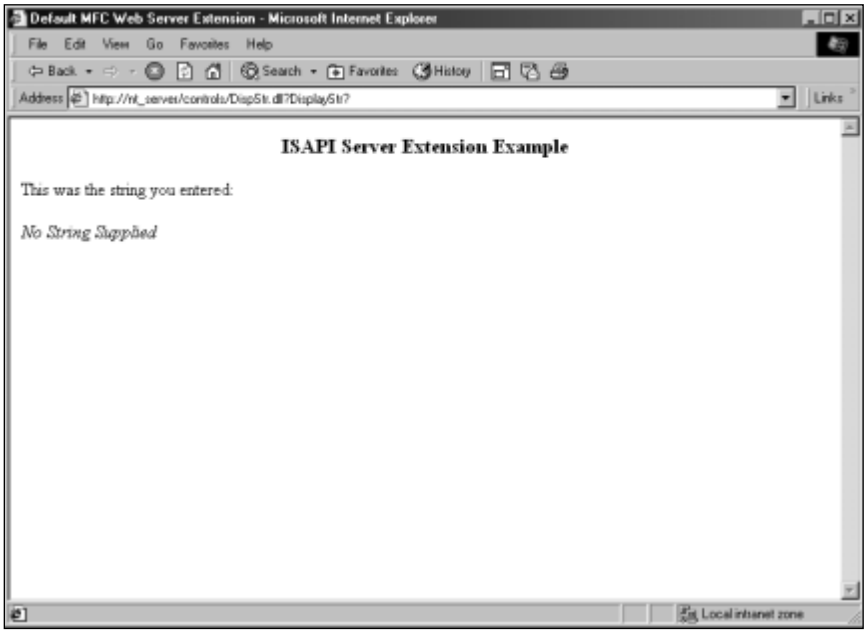

Default MFC Web Server Extension - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/controls/DispStr.dll?DisplayStr?
Links
ISAPI Server Extension Example
This was the string you entered:
No String Supplied
Local intranet zone

Figure 8-6: Calling the DispStr.DLL DisplayStr function without a string parameter displays this default string value.

Click the Back button again to get back to the Extend.HTM display. This time, type **Hello+World** into the Type a String Value field. Click the Display String button and you'll see something like the display in Figure 8-7. In this case, we asked to use the DispStr.DLL DisplayStr function and provided it with "Hello+World" as a string value.

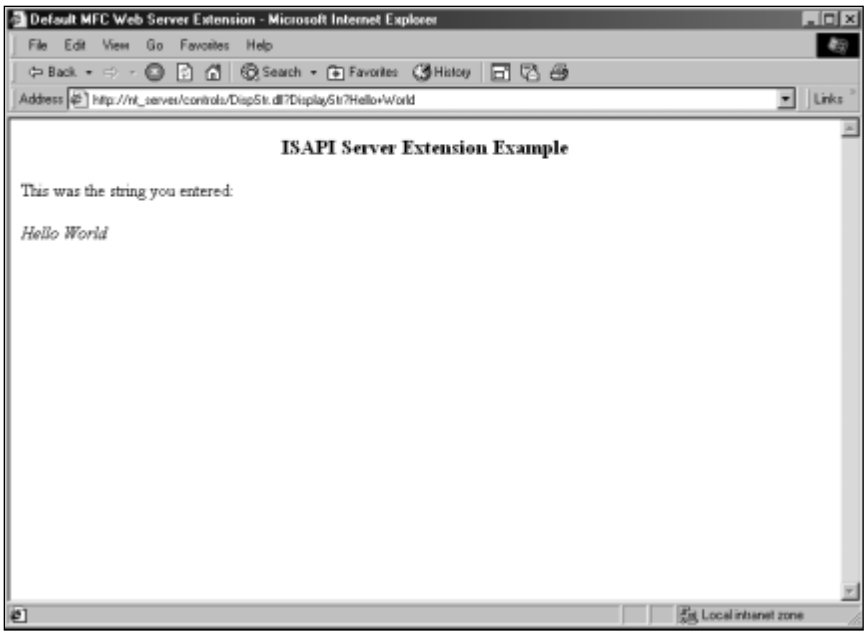

Default MFC Web Server Extension - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/controls/DispStr.dll?DisplayStr?Hello+World
Links
ISAPI Server Extension Example
This was the string you entered:
Hello World
Local intranet zone

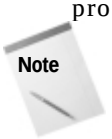
Figure 8-7: This is what happens when you provide both a function name and the required string parameter.

It's important to note that you must supply the plus sign between words when sending a string value to the server; otherwise, only the first word will get displayed. (You may want to experiment a little to see this for yourself — make sure you check the Web-server logs as you try various combinations.) If you look at the Location field on the browser, you'll notice that we have supplied a + sign between words, yet the content of the page shows a space between the words. This is because the Web server replaces the plus signs with spaces for you — this is why we didn't have to add any additional code to the ISA.

Working with Standalone ISAPI Extensions

The first example in this chapter was a simple form of standalone ISAPI Extension — it didn't rely on any scripting to produce the final Web page. Standalone ISAPI Extensions present the final result from a user request as a self-supporting Web page. In most cases, you'll use this type of ISAPI Extension to support database queries or other forms of simple informational screens. The reason you would use an ISAPI Extension instead of a script is that the ISAPI Extension can perform complex database queries faster and with less memory. In

addition, the ISAPI Extension can format the page as required for each user, help detect possible security problems, and work with a broad range of data-source products (including databases and spreadsheets).



This example assumes that you have at least a working knowledge of databases. The database itself is supplied with the source code for the book, as are the graphics files that we'll work with in this section. We won't delve into any database theory, except what's required to make the example work with the Web server. The whole purpose of this example is to show a standalone ISAPI Extension that works with database files in action.

This section of the chapter is going to show you a relatively simple database example that uses an ISAPI Extension to fetch the data. We'll use a Microsoft Access database as a data source. In our example, the database contains the inventory for a store. We're interested in displaying the list of garden implements available at the store for customers online. Of course, this same display could be used for salespersons on the road or for a variety of other purposes. (Obviously, you would need to modify the functionality of the ISAPI Extension to meet the needs of the group using the table — we'll assume a read-only application in this case.) Table 8-2 shows the layout of the Implement table of the Garden database.

Table 8-2
Garden.MDB Implement Table Layout

Field Name	*Type*	*Size*	*Format*	*Decimal*	*Description*
StockNumber	Text	50	N/A	N/A	Implement stock number.
Name	Text	50	N/A	N/A	Name of gardening implement.
NumberLeft	Number	Integer	N/A	0	Number left in stock.
Updated	Date/Time	N/A	Long Date	N/A	Date last updated in the database.
Description	Memo	N/A	N/A	N/A	Implement description.
Cost	Currency	N/A	Currency	2	How much does it cost?
ShipIt	Yes/No	N/A	Yes/No	N/A	Can we ship it?
Picture	Text	50	N/A	N/A	Location of picture.

Normally, you'd use an OLE Object data type for the Picture field. That works fine if you're going to access the database locally or over a local network connection, but because DCOM is relatively new, you can't rely on creating a network OLE connection to your data. In addition, working with BLOBs on a Web page presents certain programming challenges that most programmers won't want to take considering other methods can be used to achieve the same goal. An alternative to an OLE Object data type is the new Hyperlink data type for Office 97 and Office 2000. Unfortunately, this type of link doesn't work especially well when used locally. In addition, you'd have to have one entry for local testing and another for Web use. The use of a simple text field ends up being the best method for both local and Internet graphics work. You can programmatically use the file name stored in the field to send graphic content to the user's browser or local screen.

As you can see, this is a relatively simple database that you can implement using any number of situations (which is one of the reasons I chose it). However, this also represents a real-world use of the standalone ISAPI Extension (albeit a simple case of that real-world example). Presenting data for the user over an Internet connection is one of the most common applications that you'll find today, which is hardly surprising since it's also one of the most common business applications.

You need to take certain steps to set your Web server up for Internet use. The following list isn't all-inclusive, but it does provide you with the basics. We'll follow a simplified form of these steps:

1. Create a database. You have to build the tables, queries, and indexes required to view and manipulate your data. Make sure you test everything locally before you attempt to use it on the Internet.
2. Build an ODBC connection for your database, if required. The kind of connection you need depends on the type of database manager you're using and the ultimate goal of the database application that you're creating. In many cases, ODBC will work just fine for every type of database display and most types of data entry. I'll show you how to create the connection required for this example (assuming you retain the Access database). It's relatively easy to do, but very important. You'll need an ODBC connection to access your data through an Internet connection using ISAPI.
3. Design a program to test your database locally. This doesn't have to be anything fancy, but you do want to test your program logic for accessing the database locally before you attempt to use it remotely. At least part of this code will go right into your ISAPI Extension.
4. Create a simple ISAPI Extension. The first extension you create will simply test your ability to access a record through an Internet connection. Creating a good connection is the prime consideration. Once you have a good connection, you can start to look at other things such as displaying the information in a pleasing way and allowing the user to see more than the first record.

5. Design a Web page to view your database. There are all kinds of considerations here. For example, you may simply want a search-engine-type interface that displays information in columns. On the other hand, you may want to do something more complex.
6. Build the final ISAPI Extension. Now that you've constructed all the pieces, it's time to put them together. Obviously, at this point, you'll still need to perform some level of testing to ensure that what you have really works and that it displays the data in the correct way.

CD-ROM Location: Chapt08\DBISA1
Program Name: DBISA1.DLL
Source Code Modules in Text: DBISA1.CPP, DBISA1.H

Building an ODBC Connection

Creating a database gives you a storage container for your data, but it doesn't do much in the way of access. Sure, you can use the database engine to modify the data, but that won't work very well over the Internet. In fact, unless you were very careful in setting things up, it would probably give too much power to a user whom you only wanted to view the data. What you really need is some way to move the data from the database and display it in a regular program.

In this section, I assume that you're using a minimum of IIS version 4 installed on Windows NT Server 4 with Service Pack 3 installed. The latest version of IIS and Windows 2000 Server is the preferred configuration. If you're using any other configuration, you may need to change the procedures in this section. In addition, the screenshots shown in this chapter will differ from the ones you see.

There are two common methods you can use to look at the data in an Access database. The first method is to use Data Access Objects (DAO). Unfortunately, though this method is rather straightforward and doesn't require much setup on your part, it's limited to local and network access for the most part. (Some people have managed to set up remote links, but this kind of connection is shaky at best and probably not worth your time investigating.)

Open Database Connectivity (ODBC) is the second method to use to view your data. It has the advantage of allowing remote connections. You'll find that using ODBC does require a little additional setup on your part and that accessing the data isn't quite as straightforward.

Older versions of the ODBC drivers are not thread-safe. In other words, they could fail when used within an individual IIS execution thread. Every connection from a client to the server is executed in a thread, so using a thread-safe driver is absolutely essential. Failure to use thread-safe drivers could result in data loss. Make sure you look at the ODBC Drivers tab of the ODBC Data Source Administrator dialog to make sure all of the ODBC drivers you intend to use are version 3.50 or above. You can download the latest version of many drivers along

with the OLE DB SDK from `http://www.microsoft.com/data/oledb/download.htm#SDK`.

The first thing we'll need to do to set up ODBC is create a connection to the database. Do this using the ODBC or ODBC 32 applet in the Control Panel. Access 97 normally uses the ODBC 32 applet because it's a 32-bit database engine; however, in some cases, it appears in the standard ODBC applet (which is normally reserved for 16-bit database engines). Make sure you look for your database engine in the right applet. Figure 8-8 shows a typical example of what you'll see. Since Microsoft has changed the appearance of the ODBC Data Source Administrator quite a few times and it's unlikely that you have precisely the same database engines installed that I do, your dialog box may look different from the one shown here. The following procedure will take you through the steps required to create your own data source.

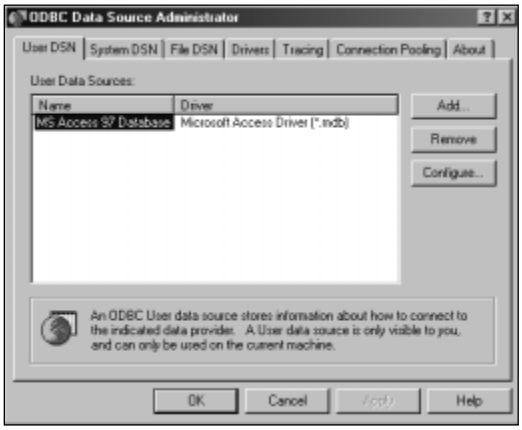


Figure 8-8: The ODBC Data Source Administrator User DSN tab contains a list of data sources you can choose from.

Try This

1. Select the System DSN tab of the ODBC Data Source Administrator dialog. Data sources that you want to access from the Internet must appear in the System DSN tab, not the User DSN or File DSN tabs.

Never use the same DSN name for entries on multiple ODBC Data Source Administrator dialog tabs. For example, if you have a DSN entry called Gardening on the System DSN tab, then you can't use that name on the User DSN tab as well. Doing so can cause the ODBC driver to return error messages. The precise error message you get depends on the ODBC driver that you're using. In most cases, the error message won't tell you that there's a duplicate entry, it will simply say that the driver couldn't access the database. For example, the Microsoft Access driver returns an error message indicating that there's some type of corruption in the registry.

2. Click Add and you'll see the Create New Data Source dialog box shown in Figure 8-9. This dialog shows the data source types that your database engine can support for the current setup. It also includes the version number of the database engine driver, the company that created the driver, the name of the driver file, and the date that the file was last updated. All of this information is important because you want to ensure you have the most up-to-date driver on your machine. Using an old driver could result in unexpected problems, most of which will be disguised as something you can't easily recognize.

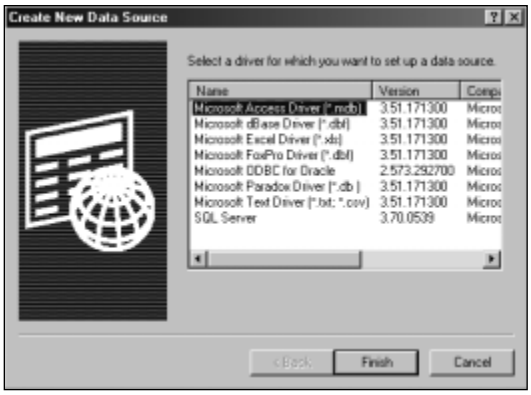


Figure 8-9: The Create New Data Source dialog box allows you to choose a database engine driver.

If you don't see a driver that you need in the Create New Data Source dialog box, it most likely means that you didn't install the required driver during database engine installation. It could also mean that the driver is corrupt or that Windows isn't recognizing it for some reason. In most cases, you'll first want to try reinstalling the driver. If it still doesn't appear in the list, contact the database engine company to see if they have an updated driver. In many cases, you'll be able to download the driver directly from the Internet. Make sure you check for third-party driver support too. More than a few companies produce specialty drivers for existing database engines.

3. Highlight the name of the database driver that you want to use. I'll be using the Microsoft Access Driver (*.MDB) for this example, but you can use any driver needed to access your particular database file type.

4. Click Finish. If you're using Microsoft Access as your database engine and the Microsoft Access Driver, then you'll see the ODBC Microsoft Access 97 Setup dialog. Every database driver has its own setup screen, which is partly driven by the type of data you need to access. Figure 8-10 shows the dialog I'll be using for this example.

From this point on, I assume that you're using Microsoft Access 97 or Access 2000 as your database engine and that you want to access a MDB file. Other database engine and driver combinations will produce similar, though different, dialogs. Make sure you read the vendor directions for setting up these other DSNs (data source names).

Figure 8-10: Microsoft Access provides this ODBC Microsoft Access 97 Setup dialog box for defining DSNs

5. Type a Data Source Name (DSN). I used Gardening for my Data Source Name field entry. The DSN you choose is important because it's the name you'll use to access the database from a Web page or within an application.
6. Type a description of your database in the Description field. I used "Contains a complete list of our gardening implement inventory." This description will appear in the ODBC Data Source Administrator dialog later, so it's important that you provide a short, accurate description of what the database contains.
7. Click the Select button. You'll see the Select Database dialog box shown in Figure 8-11. This is where you'll create the connection between the DSN entry and the actual database.

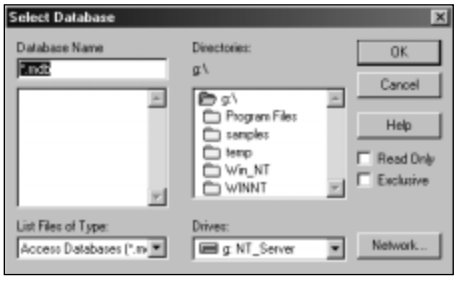


Figure 8-11: The Select Database dialog box allows you to create a connection between the DSN and your database.

8. Highlight the database you want to use (Garden.mdb in the case of the example), and then click OK. You'll return to the ODBC Microsoft Access 97 Setup dialog. The fully qualified path along with the name of your database file should appear in the Database group of the dialog box. Since we're creating a DSN for viewing purposes only, we only need to make a few other changes.
9. Click the Options button. The ODBC Microsoft Access 97 Setup dialog will extend as shown in Figure 8-12. Notice that there's an option for setting the database access to read-only. This option will change the way the DSN accesses the database, making it impossible for anyone to change its contents (at least when accessed through the DSN). The Exclusive option will open the database in exclusive mode, which means that no one else will be able to access it at all. You can use this mode when you want to perform database maintenance.

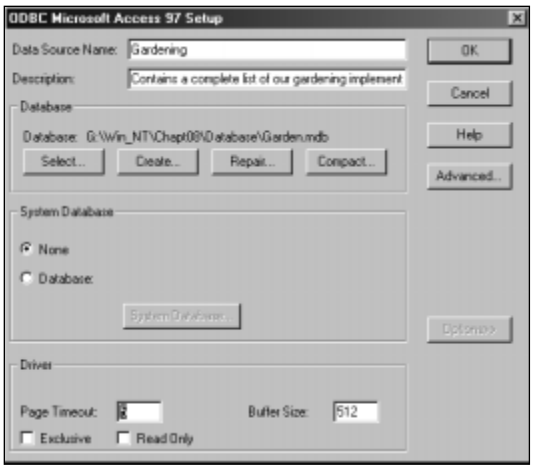


Figure 8-12: It always pays to check which options your database engine provides.

10. Click the Read-Only checkbox. This will allow us to open the database for reading only. There is one last thing you need to check.
11. Click the Advanced button. You'll see the Set Advanced Options dialog box shown in Figure 8-13. This dialog box will normally contain a lot of driver-specific entries that you can modify to improve performance to set up a specific set of conditions. However, these options are different for every database driver. Notice the Login name and Password fields. You'll find them provided for every driver that implements some type of security. If we had added security to our database earlier, you would need to add a login name and password here to provide even basic access. Remember to add security when creating a database connection for a production environment.

Figure 8-13: The Advanced Options dialog box could contain any number of entries, but the Login name and Password fields are the most common.

12. Click OK to close the Set Advanced Options dialog box.
13. Click OK to close the ODBC Microsoft Access 97 or Access 2000 Setup dialog box. You'll see the Gardening DSN added to the System DSN page of the ODBC Data Source Administrator dialog. Seeing this entry means that you now have a connection to the database that you can access from a Web page or an application.
14. Click OK to close the ODBC Data Source Administrator dialog box.

Creating a Simple ISAPI Extension

The first thing you need to do for this example is create an ISAPI Extension project like the one in the "Writing a Simple ISAPI Extension" section of this chapter. The only thing you need to know about creating the ISAPI Extension shell is that I'll be calling my project DBISA1. The following sections will take you through the same three coding steps that we followed in the previous section.

You can debug your ISAPI Extension in a number of ways, including a documented remote debugging method provided by Microsoft, but I usually go the easy route and debug the code in an application first, then transfer it over to the extension. Not only do you get full debugger capability this way, but you can use the application to give nonprogrammers an idea of what the extension can do. Unfortunately, the easy method doesn't always work. There are times when you have to debug the ISAPI code on the server and in a running environment. One of the techniques you can use to get the debugger going in this situation is to place an __asm int 3 call in your code. This will force an exception on the server and allow you to start the debugger. Obviously, you'll have to install Visual C++ on the server for this to work.

Defining the DBISA1 Parse Map

From a coding perspective, many of the things we'll do in this section will look the same as our first example. As with that first example, we need to create a parse map that points to our database function in the DBISA1.CPP file. Listing 8-6 shows the code that you'll need to add for this example.

Listing 8-6: DBISA1 Parse Map Source Code

```
////////////////////////////////////////////////////////////////////
// command-parsing map

BEGIN_PARSE_MAP(CDBISA1Extension, CHttpServer)
     // Create the parse map for our function.
     ON_PARSE_COMMAND(DispDB, CDBISA1Extension, ITS_EMPTY)

     // Default parse map created by MFC.
     ON_PARSE_COMMAND(Default, CDBISA1Extension, ITS_EMPTY)
     DEFAULT_PARSE_COMMAND(Default, CDBISA1Extension)
END_PARSE_MAP(CDBISA1Extension)
```

Notice that the DispDB( ) function won't require any input parameters. We still need to use the ITS_EMPTY argument to show that there aren't any parameters needed. As usual, the Default( ) function will contains usage instructions for this ISAPI Extension, so we'll keep it as the default command that gets called for a client request.

Declaring the DispDB( ) and WriteTitle( ) Functions

The next step will look familiar as well. We need to declare the new ISAPI Extension function as shown in Listing 8-7 in the DBISA1.H file. In this case, we're adding a WriteTitle( ) function that allows the ISAPI Extension to create a little more custom output than in the previous example. We'll add other custom features as the example progresses as well, but this is the only feature that requires us to write a special implementation function.

Listing 8-7: DispDB( ) Function Declaration Source Code

```
public:
     CDBISA1Extension();
     ~CDBISA1Extension();

// Overrides
```

```
      // ClassWizard generated virtual function overrides
              // NOTE - the ClassWizard will add and remove member functions
here.
              //    DO NOT EDIT what you see in these blocks of generated code !
      //{{AFX_VIRTUAL(CDBISA1Extension)
      public:
      virtual BOOL GetExtensionVersion(HSE_VERSION_INFO* pVer);
      //}}AFX_VIRTUAL


      // ISAPI Extension functions called by client.
      void Default(CHttpServerContext* pCtxt);
      void DispDB(CHttpServerContext* pCtxt);

    DECLARE_PARSE_MAP()

    //{{AFX_MSG(CDBISA1Extension)
    //}}AFX_MSG

protected:
    // Create a special title display function.
    void WriteTitle(CHttpServerContext* pCtxt);
```

Unlike the previous example, the only thing we need to do, in this case, is create a function that accepts the context passed to it by the server. Remember that the context determines which client request you're actually working with. Several clients could request services from the same ISAPI Extension function at the same time, so keeping thread safety in mind is essential.

The WriteTitle( ) function gets declared as a protected function since the outside world doesn't require access to it. Theoretically, you can implement this function any way you wish.

Writing the DispDB( ) and WriteTitle( ) Functions

The example we're working with this time is a little more complex than the previous one, so there is more to do when it comes to creating the example code. Since this is a database application and we didn't add support using the ISAPI Extension Wizard (it's not an option), the first task is to add database support. Listing 8-8 shows the one line of code you need to add to the StdAfx.H file to gain database support. Once you add this #include directive, you should be able to access any required database functions and classes.

Listing 8-8: **Including Database Support**

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#include <afx.h>
#include <afxwin.h>
#include <afxmt.h>           // for synchronization objects
#include <afxext.h>
#include <afxisapi.h>
#include <afxdb.h>           // Added database support.
```

Writing the DispDB( ) function comes next. Listing 8-9 contains the complete source for this function. Don't worry about the length of the code, we'll talk about how it works once you've had a chance to look through it.

Listing 8-9: **void CDBISA1Extension::DispDB(CHttpServer Context* pCtxt)**

```
{
    // Create some variables for database access.

    CDatabase            oDB;        // Database object.
    CRecordset            oRS;        // Recordset object.
    CCriticalSection    oCritical;    // Critical section for database
operations.

     CString oField; // Used to hold field values.

     // Display a heading.
     StartContent(pCtxt);
     WriteTitle(pCtxt);

     // Lock the critical section so that other users can't attempt to
     // open the database.
     oCritical.Lock();

     try
     {
             // Open the database.
             oDB.OpenEx(_T("DSN=Gardening"),
                     CDatabase::openReadOnly | CDatabase::noOdbcDialog);
```

```
            // Assign the database to our recordset
            //(allows use of the database connection).
            oRS.m_pDatabase = &oDB;

            // Open the recordset.
            oRS.Open(CRecordset::snapshot,
                    _T("SELECT StockNumber, Name, NumberLeft, Updated, Cost,
Picture FROM Implement ORDER BY Name"),
                    CRecordset::none);
     }
     catch (CDBException* e)
     {
            *pCtxt << _T("\n<H2><CENTER>Database Opening
Error</H2></CENTER>\n\n");
            *pCtxt << _T(e->m_strError);
            *pCtxt << _T("\n<P>\n");
            *pCtxt << _T(e->m_strStateNativeOrigin);

            // Unlock the critical section.
            oCritical.Unlock();

            // End the document.
            EndContent(pCtxt);

            // Exit the ISAPI Extension
            return;
     }

     // Unlock the critical section now that we're done with the database.
     oCritical.Unlock();

     // Create a heading.
     *pCtxt << _T("\n<H2><CENTER>ISAPI Database Access
Example</H2></CENTER>\n\n");

     // Start a table.
     *pCtxt << _T("<CENTER>\n<TABLE WIDTH = 95% BORDER = 2>\n\n");
     *pCtxt << _T("<TR>\n");
     *pCtxt << _T("<TH>Name</TH>\n");
     *pCtxt << _T("<TH>Stock #</TH>\n");
     *pCtxt << _T("<TH>Qty</TH>\n");
     *pCtxt << _T("<TH>Cost</TH>\n");
     *pCtxt << _T("<TH>Updated</TH> \n");
     *pCtxt << _T("<TH>Picture</TH>\n\n");

     // Output the contents of the database.
     try
     {
            while (!oRS.IsEOF())
            {
                    // Start a row.
                    *pCtxt << _T("<TR>\n");
```

Continued

Listing 8-9: **(***continued)*

```
                    // Display the Name field.
                    *pCtxt << _T("<TD>");
                    oRS.GetFieldValue("Name", oField);
                    *pCtxt << _T(oField);
                    *pCtxt << _T("</TD>\n");

                    // Display the Stock Number field.
                    *pCtxt << _T("<TD>");
                    oRS.GetFieldValue("StockNumber", oField);
                    *pCtxt << _T(oField);
                    *pCtxt << _T("</TD>\n");

                    // Display the NumberLeft field.
                    *pCtxt << _T("<TD>");
                    oRS.GetFieldValue("NumberLeft", oField);
                    *pCtxt << _T(oField);
                    *pCtxt << _T("</TD>\n");

                    // Display the Cost field.
                    *pCtxt << _T("<TD ALIGN=RIGHT>$");
                    oRS.GetFieldValue("Cost", oField);
                    *pCtxt << _T(oField.Left(oField.GetLength() - 2));
                    *pCtxt << _T("</TD>\n");

                    // Display the Updated field.
                    *pCtxt << _T("<TD>");
                    oRS.GetFieldValue("Updated", oField);
                    *pCtxt << _T(oField.Left(10));
                    *pCtxt << _T("</TD>\n");

                    // Display the Picture field.
                    *pCtxt << _T("<TD ALIGN=CENTER><IMG
SRC=http://NT_Server/Files/");
                    oRS.GetFieldValue("Picture", oField);
                    *pCtxt << _T(oField);
                    *pCtxt << _T("></TD>\n\n");

                    // Move to the next record.
                    oRS.MoveNext();
            }
     }
     catch (CDBException* e)
     {
            *pCtxt << _T("\n<H2><CENTER>Database Data Output
Error</H2></CENTER>\n\n");
            *pCtxt << _T(e->m_strError);
            *pCtxt << _T("\n<P>\n");
            *pCtxt << _T(e->m_strStateNativeOrigin);
     }

     // End the table.
```

```
        *pCtxt << _T("</TABLE>\n</CENTER>\n\n");

        // Declare a critical section before closing the database and recordset.
        oCritical.Lock();

        try
        {

                // Close the recordset.
                if (oRS.IsOpen())
                        oRS.Close();

                // Close the database.
                if (oDB.IsOpen())
                        oDB.Close();
        }
        catch (CDBException* e)
        {
                *pCtxt << _T("\n<H2><CENTER>Database Closing
Error</H2></CENTER>\n\n");
                *pCtxt << _T(e->m_strError);
                *pCtxt << _T("\n<P>\n");
                *pCtxt << _T(e->m_strStateNativeOrigin);
        }

        // Unlock the critical section.
        oCritical.Unlock();

        // End the document.
        EndContent(pCtxt);
}
```

Some of the changes in this example aren't much different from those in the previous example. For example, you still need to include code for creating a Web page header and for ending the Web page once the content is complete. You'll find this particular code in any ISAPI Extension that you create.

However, once we get past these common elements in the source file shown in Listing 8-9, things change a great deal. The first thing you should see are three local variable declarations near the top of the DispDB( ) function. oDB is the database object, oRS is the recordset object, and oCritical is a critical section object. You need all three of these variables to access the database and the tables it contains. The oField variable will eventually hold database data.

There are situations when you'd make the three database variables global instead of local to the database function. For example, you may want to create more than one database display function or multiple functions may need to access the same database. However, remember that these functions need to be thread-safe, which means that using global variables could be a cause for problems. It's always better to use local variables if you can.

The oCritical.Lock( ) begins a critical section. A critical section tells IIS that you don't want more than one thread executing the code the follows at one time. Even though our ODBC driver is supposedly thread-safe, using a critical section reduces the chance that a problem will occur while opening the database. Microsoft recommends you take this step. Using a critical section also ensures that you can at least detect errors encountered in opening the database for use, an important consideration when the user may not be able to contact anyone for help. Opening a database within a critical section means that you'll be able to read the error without any possibility of corruption from another process. In other words, you'll want to use critical sections for more than database work; they're important whenever you want to ensure an ISAPI Extension's access to a particular resource.

The use of critical sections has some negative points to consider as well. The most important thing to remember is that you have only one thread of execution within a critical section. Using a single thread of execution means users could get blocked while waiting for access. You want to make critical sections as small as possible to reduce the amount of drag on the system and the amount of time an individual user can expect to wait for access to a resource such as a database.

We're not done with the safety element of accessing the database either. Notice that the very next section of code is within a try...catch structure. You'll want to use a try...catch structure like the one shown for any type of critical access. Database access can get blocked in a lot of different ways. The error messages you receive from this structure will contain the only clues you'll get as to the source of the problem. This structure serves another important purpose as well. If you don't catch exceptions, in this case, all the user will see is a message saying that the page doesn't contain any data because the ISAPI Extension will terminate prematurely. Catching the exception at least gives you the chance to provide the user with some type of output. Notice that the catch part of the structure actually sends the error information to the user's browser, so that the error can be documented. (Once you have sufficient information from the user to duplicate the problem, you can also use the ODBC Trace feature to provide yourself with detailed logs of exactly what's going on with the database connection.)

At this point, we can finally attempt to open the database. The database opening sequence is the three-step process shown below:

1. Open the database using the OpenEx( ) function. Make sure you include a DSN that appears on the System DSN tab of the ODBC Data Source Administrator dialog. (You can access this dialog through the ODBC applet in the Control Panel.) You must also use the CDatabase::noOdbcDialog option to ensure that the ODBC driver doesn't attempt to present the user with an open dialog box. Since the user is using a remote connection over the Internet, the ODBC open dialog will always cause an exception, even if everything else is set up correctly on your server.

2. Assign the database connection to the recordset object. You'll see an interesting error message if you fail to perform this step. The recordset object will assume that you simply failed to create a database object and ask the ODBC driver to display the ODBC open dialog. You'll get an error message that looks exactly the same as if you forget to include the CDatabase::noOdbcDialog option with the OpenEx( ) function in the previous step.
3. Open the recordset using the Open( ) function. You can do this in a variety of ways. The example code shows how you can use a SQL query, which you could pass as part of the request when using a generic ISAPI Extension. Other options include using a query or asking for a specific table. Under no circumstances should you try to open a recordset without specifying the data you want to see. The results will be an exception with an error message that will have you running in circles for hours. For some reason, none of the ODBC drivers I tried provide a clear message of what's wrong if you fail to provide a specific information request.

Now that we have the database open, we can unlock the critical section. This tells IIS that it's safe to allow other threads to use the code in the critical section. Make sure you unlock the critical section as soon as it's safe to do so. Otherwise, you'll end up making other users wait unnecessarily.

It's time to display some data. You can display the data in any number of formats, but a table works well if you have a limited amount of data to show. The example code shows how you can put a table together for display purposes. Notice that the code begins by displaying title and column headings for the table. You use the <TH> tag to differentiate table headings from table data. In addition, each row begins with the <TR> tag.

Displaying data in your table is a four-step process:

1. Display the beginning tag <TD>. Notice that the code uses the ALIGN attribute to change the alignment of individual columns. The default alignment is left.
2. Get the data from the database using the GetFieldValue( ) function. There are actually several different forms of this particular function. The form shown in the sample code requests data from a specific field and places it in a CString object. Make sure you use the method most appropriate for the type of data you want to display.
3. Output the data. It's when you output the data that you begin to notice some limitations of the CString method used in the example code. For one thing, you have no control over the appearance of dates. Another problem is that you have to rely on the Left( ) and Right( ) functions to remove excess information from the string. In other words, you can create output very quickly using the method shown, but your formatting options are somewhat limited.
4. Display the closing tag </TD>. You must include this tag or all of the data will run together into one table entry.

The URL for accessing the picture files in this example is hard coded for ease of viewing. Normally, you'd use a function to derive the server name and then add the default storage location along with a file name. Make sure you change the hard-coded portion of the code to match your server configuration before trying this example. You'll need to recompile the sample code as well before the change will take effect.

You have to use the MoveNext( ) function to get from record to record in the database. Notice that this function appears at the end of the while loop. As soon as the ISAPI Extension reaches the last record, the IsEOF( ) function will return true, ending the loop.

The last part of the standard content for our Web page is the </TABLE> required to end the table. Since the table was centered on the browser window, we also need to include the </CENTER> tag. Make sure you place these tags in the reverse order in which they appear at the beginning of the table. Most browsers don't make a differentiation, but some do.

Closing the recordset and database comes next. Notice that we create another critical section to ensure that the database gets closed as expected. We also use a similar try...catch structure to ensure that any exceptions get trapped and displayed on the user's browser. It's just as important to make sure the database gets closed properly as it is to open it properly.

The very last step in the whole display process is to unlock the critical section. Even though the user won't see anything, the server will behave strangely if you don't perform this final critical section unlock.

We still have a WriteTitle( ) function to think about. Remember that the WriteTitle( ) function creates the entire heading areas of the final Web page, not just the <TITLE> tag. Listing 8-10 shows the code for this function.

Listing 8-10: **WriteTitle( ) Function Source Code**

```
void CDBISA1Extension::WriteTitle(CHttpServerContext* pCtxt)
{
    // Create a standard heading.
    *pCtxt << _T("<HTML>\r\n<HEAD>\r\n");

    // Display a title
    *pCtxt << _T("<TITLE>A Standalone Database Example</TITLE>\r\n");

    // Start the body.
    *pCtxt << _T("</HEAD>\r\n<BODY>\r\n");
}
```

As you can see, this function is relatively simple. There are three output lines: heading, title, and start of body. These are the same tags that you would use when creating any Web page. Notice that use of the \r\n escape sequence within the code. This sequence actually places returns on the resulting Web page source code, making it easier to troubleshoot.

Obviously this is a very simple WriteTitle( ) implementation. You can use this function to add scripts and other features to the Web page that normally get placed in the header area. We also haven't defined any <META> tags or any of the other tags that normally appear within a Web page.

Designing a Test Web Page

Now that we have an ISAPI Extension for displaying the contents of our sample database, it's time to create a Web page for accessing it. In most cases, you protect this page by requiring a password to access it. For the purposes of this example, you probably won't need to go to that extreme. Hopefully you're using a test server that no one else can access. Listing 8-11 shows the code for this example. We'll discuss it after you get a chance to view the code.

CD-ROM Location: Chapt08\DBISATst
Program Name: DBISATst.HTM
Source Code Modules in Text: DBISATst.HTM

Listing 8-11: **Test Web Page Source Code**

```
<HTML>
<HEAD>
<TITLE>Gardening Database Access Test Page</TITLE>
</HEAD>
<FORM>

<!- Create a heading. ->
<CENTER><H2>Check Out Our Garden Inventory</H2></CENTER>

<!- Display an access message and button. ->
Click the button to see the <STRONG>Inventory</STRONG>.<P>

<INPUT LANGUAGE="JavaScript"
    TYPE=BUTTON
    VALUE="Take a Look"
    ONCLICK="window.location.href =
'http://nt_server/controls/DBISA1.DLL?DispDB'">
```

Continued

Listing 8-11: *(continued)*

```
<P>
<INPUT LANGUAGE="JavaScript"
    TYPE=BUTTON
    VALUE="Check Default"
    ONCLICK="window.location.href =
'http://nt_server/controls/DBISA1.DLL?Default'">

</FORM>
</HTML>
```

There shouldn't be any surprises with this code. The only thing we do is display a heading and a couple of buttons for accessing the DBISA1 ISAPI Extension. Figure 8-14 shows what this Web page looks like when you access it.

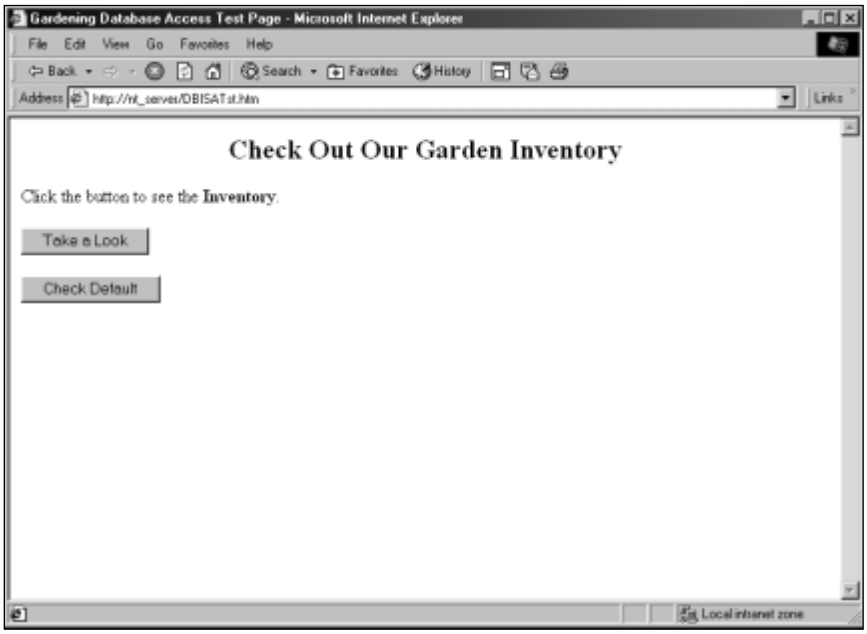

Gardening Database Access Test Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/DBISAT st.htm
Links
Check Out Our Garden Inventory
Click the button to see the Inventory.
Take a Look
Check Default
Local intranet zone

Figure 8-14: The Web page you use to access the database is normally password-protected.

Testing the Final Result

Unlike many of the other examples we've worked with so far, it's important to test your database-oriented ISAPI Extension for failure as well as success. Of course, you'll probably want to test for success first. At this point, you should have your ISAPI Extension copied to the server along with the test Web page and all of the data. You should also have created a DSN entry using the ODBC applet in the Control Panel. If you've done everything right, what you should see when you click the Take a Look button on the text Web page is the test database shown in Figure 8-15. Notice that the title we created with the WriteTitle( ) function appears on the title bar of the browser.

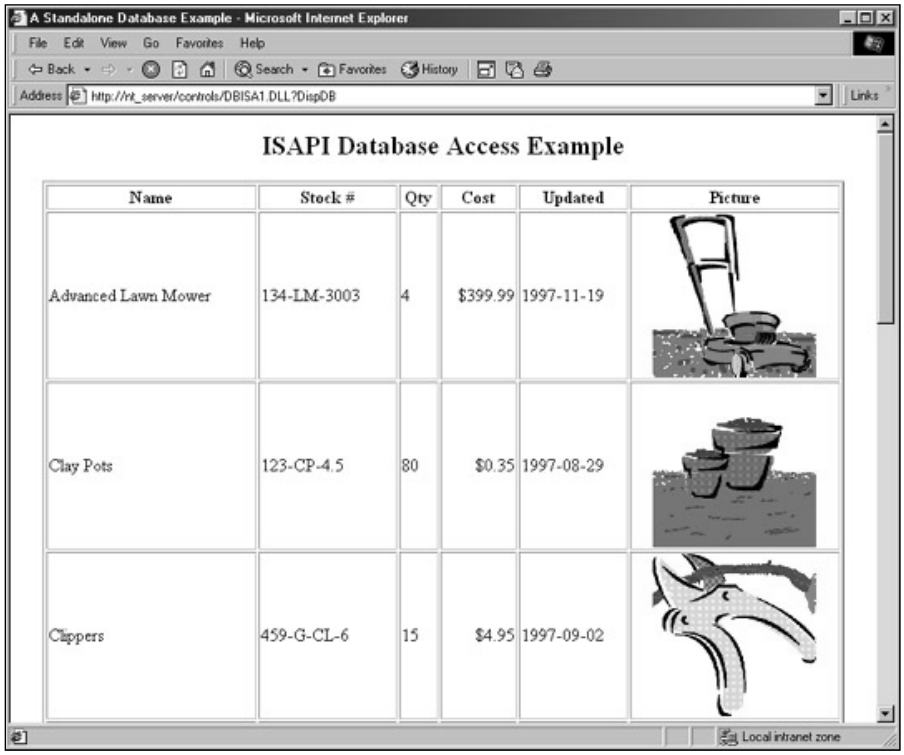


Figure 8-15: Make sure you have a working configuration before you try testing the error handling of your ISAPI Extension.

Now it's time to test for failure. One of the things you could do is remove the DSN entry from the System DSN page of the ODBC Data Source Administrator dialog. Figure 8-16 shows an example of what you'll see with this type of error. You can also

try adding the same DSN name on multiple pages of the ODBC Data Source Administrator dialog (for example, you can create a Gardening entry on both the User DSN and System DSN pages). Moving the table to another location or even creating a damaged table (after you've made a copy of the original of course) are other ways that you can check the error-detecting capability of your ISAPI Extension and ODBC driver.

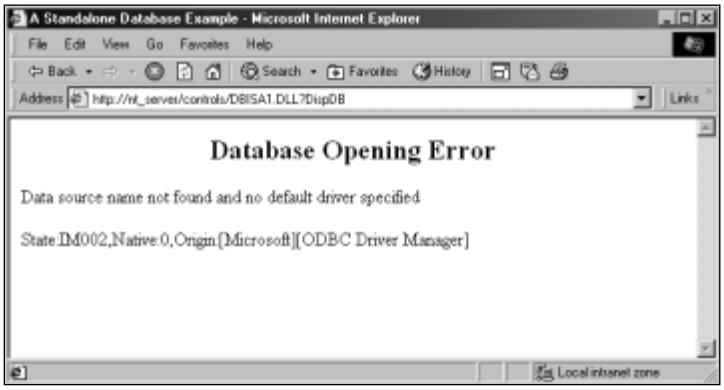

A Standalone Database Example - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/controls/DBISA1.DLL?DispDB
Links
Database Opening Error
Data source name not found and no default driver specified
State:IM002,Native:0,Origin:[Microsoft][ODBC Driver Manager]
Local intranet zone

Figure 8-16: It's important to test several failure modes as well as what you think will work.

Most browsers cache the pages they receive from the Internet to save time downloading them later. This has the unfortunate side effect of making some of your changes to an ISAPI Extension invisible. Make sure you press the Refresh or Reload button of your browser to force a reload of the page if the results you see onscreen don't match the results you were expecting.

If possible, you'll want to check online error conditions as well. For example, what effect would line noise have on data transmission? It's important to check for things that will affect the user as well as your database so that support personnel know what to look for. You also want to check to see what happens if the user disconnects suddenly. Databases that accept user input would want to check the type of information the user is providing and whether that information is what you expected to get.

Working with Script-dependent ISAPI Extensions

ISAPI Extensions and scripts aren't necessarily mutually exclusive. Sometimes you'll want the ease and lower learning curve of programming with scripts when working with less complex applications. This ease is especially important when

working with pages that contain generic material requiring a little more processing than a simple Web-page script will allow.

Client-side processing provides the greatest opportunity to use an ISAPI Extension with a script. The user can make certain choices that are then relayed to the server as part of a client request. A client-side script can preprocess the information, reducing the load on the server. Adding ISAPI DLL capabilities further reduces the server load and increases Web-site responsiveness, all without any loss of functionality or server accessibility.

Server-side scripting normally involves some type of Active Server Page (ASP) file and its associated scripting language. ASP provides formidable capabilities and many developers use ASP to the exclusion of every other technology. However, going this route does involve a cost in server performance that reduces the number of users that a single server can handle, and the responsiveness of the system as a whole. As with most scripting technologies, an ASP script has to be reinterpreted every time a user makes a request, which means that the server has to expend the processor cycles required to load the script and an interpreter. A better approach is to use ASP to create the initial Web page and any simple processing, then use an ISAPI Extension to handle redundant or complex requests. Since the ISAPI Extension remains in memory after the first client request and uses machine code, rather than an interpreter, it's much more efficient.

There are two levels of script coverage provided in this book. The first is client-side script processing provided by scripting languages such as JavaScript and VBScript. The second is server-side script processing that is performed by using ASP files and its associated scripting language. This section of the chapter deals mainly with the client-side processing issues using JavaScript. We'll look at server-side scripting using ASP in detail in Chapter 10.

So far, the chapter has shown you several instances where a script attached to a button calls an ISAPI Extension to display a Web page. We really haven't explored the option of performing client-side or server-side script processing first, then using the results as part of the calling criteria for an ISAPI Extension. That's what we'll look at in this section of the chapter — using a client-side script to feed an ISAPI Extension processed data.

CD-ROM Location: Chapt08\ShowVar
Program Name: ShowVar.DLL
Source Code Modules in Text: ShowVar.CPP

Our example program uses a client-side script to modify the output of an ISAPI Extension. However, some background processing is going on as well that the client will modify, but that the user won't be aware of. In this case, we'll use the GetServerVariable( ) method of the CHttpServerContext class to interrogate both the server and the client. We'll use input from the user to determine how the data is formatted, but the actual data will be supplied outside the user's input.

Creating the ShowVar ISAPI Extension

We've created two ISAPI Extensions in this chapter so far, so you should have a pretty good idea of how the basic ISAPI Extension creation process works. Create a new ISAPI Extension project named ShowVar. To this project add a parse map and function declaration for a function named ClientView. This function will take two string parameters named Format and Full. The Format parameter will determine the appearance of the output data, while the Full parameter will determine whether just the client or both the client and server information gets displayed. The default value for Format is "Strings"; Full has a default value of "False". Your function declaration should look like the one shown here:

```
void ClientView(CHttpServerContext* pCtxt, LPTSTR lpszFormat,
LPTSTR lpszFull);
```

Once you have the function declaration and parse map in place, you can write the function code. Listing 8-12 shows the code we'll use for this example. Notice how the code handles both the supplied data and the data that it has to derive through calls to the GetServerVariable( ) method. Theoretically, I could have divided this function into several special-purpose functions and called them as needed to keep the code cleaner. I chose the single function route to keep all of the code together for explanatory purposes. I'll explain how the code works once you get a chance to read through it.

Listing 8-12: **ClientView( ) Function Source Code**

```
void CShowVarExtension::ClientView(CHttpServerContext* pCtxt, LPTSTR lpszFormat,
LPTSTR lpszFull)
{
  CString  oAuthType;  // Authorization Type
  CString  oContLen;  // Content Length
  CString  oContType;  // Content Type
  CString  oCGI;    // Common Gateway Interface
  CString  oPathInfo;  // Path information provided by client.
  CString  oPathTrans;  // Expanded form of path information
  CString  oQueryStr;  // Query string (follows ? in URL)
  CString  oRemAddr;  // IP address of client
  CString  oRemHost;  // Host name of client
  CString  oRemUser;  // User name supplied by client
  CString oReqMeth;  // HTTP request method
  CString  oScrptName;  // Name of script or ISAPI Extension
  CString  oServName;  // IP address of server
  CString  oServPort;  // TCP/IP port that received request
  CString  oServProt;  // Name and version of retrieval protocol
  CString  oServSoft;  // Name and version of Web server software
  CString  oAllHTTP;  // Any unparsed HTTP header information
  CString  oHTTPAccpt;  // Data formats the client can accept
  DWORD  dwSize;    // The returned size of the value.
```

```
// This section of the code fills variables with the various
// client and server related values we can obtain using the
// GetServerVariable() method.

// Get the authorization type.
dwSize = 40;
pCtxt->GetServerVariable("AUTH_TYPE",
  (LPVOID)oAuthType.GetBuffer(40),
  &dwSize);
oAuthType.ReleaseBuffer(-1);

// Get the content length.
dwSize = 40;
pCtxt->GetServerVariable("CONTENT_LENGTH",
  (LPVOID)oContLen.GetBuffer(40),
  &dwSize);
oContLen.ReleaseBuffer(-1);

// Get the content type.
dwSize = 40;
pCtxt->GetServerVariable("CONTENT_TYPE",
  (LPVOID)oContType.GetBuffer(40),
  &dwSize);
oContType.ReleaseBuffer(-1);

// Get the gateway interface.
dwSize = 40;
pCtxt->GetServerVariable("GATEWAY_INTERFACE",
  (LPVOID)oCGI.GetBuffer(40),
  &dwSize);
oCGI.ReleaseBuffer(-1);

// Get the path information.
dwSize = 40;
pCtxt->GetServerVariable("PATH_INFO",
  (LPVOID)oPathInfo.GetBuffer(40),
  &dwSize);
oPathInfo.ReleaseBuffer(-1);

// Get the translated path.
dwSize = 40;
pCtxt->GetServerVariable("PATH_TRANSLATED",
  (LPVOID)oPathTrans.GetBuffer(40),
  &dwSize);
oPathTrans.ReleaseBuffer(-1);

// Get the query string.
dwSize = 40;
pCtxt->GetServerVariable("QUERY_STRING",
  (LPVOID)oQueryStr.GetBuffer(40),
  &dwSize);
oQueryStr.ReleaseBuffer(-1);
```

Continued

Listing 8-12: *(continued)*

```
// Get the remote address.
dwSize = 40;
pCtxt->GetServerVariable("REMOTE_ADDR",
  (LPVOID)oRemAddr.GetBuffer(40),
  &dwSize);
oRemAddr.ReleaseBuffer(-1);

// Get the remote host.
dwSize = 40;
pCtxt->GetServerVariable("REMOTE_HOST",
  (LPVOID)oRemHost.GetBuffer(40),
  &dwSize);
oRemHost.ReleaseBuffer(-1);

// Get the remote user.
dwSize = 40;
pCtxt->GetServerVariable("REMOTE_USER",
  (LPVOID)oRemUser.GetBuffer(40),
  &dwSize);
oRemUser.ReleaseBuffer(-1);

// Get the request method.
dwSize = 40;
pCtxt->GetServerVariable("REQUEST_METHOD",
  (LPVOID)oReqMeth.GetBuffer(40),
  &dwSize);
oReqMeth.ReleaseBuffer(-1);

// Get the script name.
dwSize = 40;
pCtxt->GetServerVariable("SCRIPT_NAME",
  (LPVOID)oScrptName.GetBuffer(40),
  &dwSize);
oScrptName.ReleaseBuffer(-1);

// Get the server name.
dwSize = 40;
pCtxt->GetServerVariable("SERVER_NAME",
  (LPVOID)oServName.GetBuffer(40),
  &dwSize);
oServName.ReleaseBuffer(-1);

// Get the server port.
dwSize = 40;
pCtxt->GetServerVariable("SERVER_PORT",
  (LPVOID)oServPort.GetBuffer(40),
  &dwSize);
oServPort.ReleaseBuffer(-1);

// Get the server protocol.
dwSize = 40;
```

```
pCtxt->GetServerVariable("SERVER_PROTOCOL",
  (LPVOID)oServProt.GetBuffer(40),
  &dwSize);
oServProt.ReleaseBuffer(-1);

// Get the server software.
dwSize = 40;
pCtxt->GetServerVariable("SERVER_SOFTWARE",
  (LPVOID)oServSoft.GetBuffer(40),
  &dwSize);
oServSoft.ReleaseBuffer(-1);

// Get all of the remaining HTTP header.
dwSize = 40;
pCtxt->GetServerVariable("ALL_HTTP",
  (LPVOID)oAllHTTP.GetBuffer(40),
  &dwSize);
oAllHTTP.ReleaseBuffer(-1);

// Get the HTTP acceptance types.
dwSize = 40;
pCtxt->GetServerVariable("HTTP_ACCEPT",
  (LPVOID)oHTTPAccpt.GetBuffer(40),
  &dwSize);
oHTTPAccpt.ReleaseBuffer(-1);

// Once we have the variables filled with data, we
// can create the Web page used to display them.

// Display a heading.
StartContent(pCtxt);
WriteTitle(pCtxt);

// Check which format the user wanted.
// If Strings is selected, then output the text as
// individual strings.
if (strncmp(lpszFormat, "Strings", 40) == 0)
{
  *pCtxt << _T("\r\n<CENTER><H1>String Formatted Output</H1></CENTER>");
  *pCtxt << _T("\r\n\r\n<H2>Client Entries</H2>");
  *pCtxt << _T("\r\nThe Authorization Type is: ");
  *pCtxt << _T(oAuthType.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Content Length is: ");
  *pCtxt << _T(oContLen.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Content Type is: ");
  *pCtxt << _T(oContType.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Gateway Interface is: ");
  *pCtxt << _T(oCGI.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Path Information is: ");
  *pCtxt << _T(oPathInfo.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Translated Path (directory specification) is: ");
  *pCtxt << _T(oPathTrans.GetBuffer(40));
```

Continued

Listing 8-12: *(continued)*

```
  *pCtxt << _T("<BR>\r\nThe Query String (part after the ?) is: ");
  *pCtxt << _T(oQueryStr.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Remote Address is: ");
  *pCtxt << _T(oRemAddr.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Remote Host is: ");
  *pCtxt << _T(oRemHost.GetBuffer(40));
  *pCtxt << _T("<BR>\r\nThe Remote User is: ");
  *pCtxt << _T(oRemUser.GetBuffer(40));
  *pCtxt << _T("<P>\r\n");

  // See if the user wants to see the server information.
  if (strncmp(lpszFull, "True", 40) == 0)
  {
    *pCtxt << _T("\r\n<H2>Server Entries</H2>");
    *pCtxt << _T("\r\nThe Request Method is: ");
    *pCtxt << _T(oReqMeth.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe Script Name is: ");
    *pCtxt << _T(oScrptName.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe Server Name is: ");
    *pCtxt << _T(oServName.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe Server Port is: ");
    *pCtxt << _T(oServPort.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe Server Protocol is: ");
    *pCtxt << _T(oServProt.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe Server Web Software is: ");
    *pCtxt << _T(oServSoft.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe remaining HTTP header entries are: ");
    *pCtxt << _T(oAllHTTP.GetBuffer(40));
    *pCtxt << _T("<BR>\r\nThe types of files the client can accept are: ");
    *pCtxt << _T(oHTTPAccpt.GetBuffer(40));
    *pCtxt << _T("<P>\r\n");
  }
}

// If Table is selected, then output the value name in one column,
// and the actual value in a second column.
else if(strncmp(lpszFormat, "Table", 40) == 0)
{
  // Create the table heading.
  *pCtxt << _T("\r\n<CENTER><H1>Table Formatted Output</H1></CENTER>");
  *pCtxt << _T("\r\n\r\n<CENTER><TABLE BORDER>");
  *pCtxt << _T("\r\n<CAPTION>Client Entries</CAPTION>");
  *pCtxt << _T("\r\n<TR>\r\n\t<TH>Name</TH>");
  *pCtxt << _T("\r\n\t<TH>Value</TH>");

  // Display the client data.
  *pCtxt << _T("\r\n<TR>\r\n\t<TD>Authorization Type</TD>\r\n\t<TD>");
  *pCtxt << _T(oAuthType.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Content Length</TD>\r\n\t<TD>");
  *pCtxt << _T(oContLen.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Content Type</TD>\r\n\t<TD>");
```

```
  *pCtxt << _T(oContType.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Gateway Interface</TD>\r\n\t<TD>");
  *pCtxt << _T(oCGI.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Path Information</TD>\r\n\t<TD>");
  *pCtxt << _T(oPathInfo.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Translated Path</TD>\r\n\t<TD>");
  *pCtxt << _T(oPathTrans.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Query String</TD>\r\n\t<TD>");
  *pCtxt << _T(oQueryStr.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Remote Address</TD>\r\n\t<TD>");
  *pCtxt << _T(oRemAddr.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Remote Host</TD>\r\n\t<TD>");
  *pCtxt << _T(oRemHost.GetBuffer(40));
  *pCtxt << _T("</TD>\r\n<TR>\r\n\t<TD>Remote User</TD>\r\n\t<TD>");
  *pCtxt << _T(oRemUser.GetBuffer(40));

  // See if the user wants to display the server data.
  if (strncmp(lpszFull, "True", 40) == 0)
  {
    // End the table.
    *pCtxt << _T("\r\n</TABLE></CENTER>\r\n");

    // Create the table heading.
    *pCtxt << _T("\r\n\r\n<CENTER><TABLE BORDER>");
    *pCtxt << _T("\r\n<CAPTION>Server Entries</CAPTION>");
    *pCtxt << _T("\r\n<TR>\r\n\t<TH>Name</TH>");
    *pCtxt << _T("\r\n\t<TH>Value</TH>");

    // Display the client data.
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Request Method</TD>\r\n\t<TD>");
    *pCtxt << _T(oReqMeth.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Script Name</TD>\r\n\t<TD>");
    *pCtxt << _T(oScrptName.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Server Name</TD>\r\n\t<TD>");
    *pCtxt << _T(oServName.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Server Port</TD>\r\n\t<TD>");
    *pCtxt << _T(oServPort.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Server Protocol</TD>\r\n\t<TD>");
    *pCtxt << _T(oServProt.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>Server Software</TD>\r\n\t<TD>");
    *pCtxt << _T(oServSoft.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>All HTTP</TD>\r\n\t<TD>");
    *pCtxt << _T(oAllHTTP.GetBuffer(40));
    *pCtxt << _T("\r\n<TR>\r\n\t<TD>HTTP Accept</TD>\r\n\t<TD>");
    *pCtxt << _T(oHTTPAccpt.GetBuffer(40));
  }

  // End the table.
  *pCtxt << _T("\r\n</TABLE></CENTER>\r\n");
}
// If neither Table or Strings is selected, then there
// must be an error.
```

Continued

Listing 8-12: *(continued)*

```
else
{
  *pCtxt << _T("\r\n<CENTER><H1>Error in Input</H1></CENTER>");
  *pCtxt << _T("\r\nYou've made an error in selecting the ");
  *pCtxt << _T("format for this page.  Choosed either Strings ");
  *pCtxt << _T("or Table as your format type.<P>");
  *pCtxt << _T("\r\nYou chose: ");
  *pCtxt << _T(lpszFormat);
  *pCtxt << _T(" as the format type.\r\n");
}

// End the Web page content.
EndContent(pCtxt);
 }
```

There's a lot of code in Listing 8-12, but it can easily be divided into seven major tasks:

1. Create the variables required to hold the client and server information values. You can use LPTSTR variables, but I find that CString class objects are easier to work with. If you do use a CString class object, you must observe all of the rules for working with them including releasing the buffer after creating one for use in places where you need to provide a buffer.
2. Get the client and server information using the GetServerVariable( ) method. This method requires three arguments as input: the name of the variable that you want to retrieve, a pointer to a data buffer (of type LPVOID), and the length of the buffer. Notice that the code uses the CString GetBuffer( ) method to provide a buffer for the GetServerVariable( ) method. In addition, notice that the code calls the ReleaseBuffer( ) method after the GetServerVariable( ) method call completes.
3. Create a generic header for the Web page. We've seen how to do this in every example in the chapter so far, so there isn't much need to explain these two calls again.
4. Examine the Format argument to determine if the user wants the string format, the table format, or has asked for a nonexistent format. If the user has requested a nonexistent format, the ISAPI Extension will display an error message that explains which parameters to provide for the client view method. The method defaults to a Strings display without server information.
5. If the user selected the Strings format, output the required HTML tags for the client section. There are two phases for creating the output. The first phase is to output the client data (including an identifying header). Once the client data is output, examine the Full argument to see if the user also wants to output the server data. If so, output the server data, including an identifying header. (We'll look at the actual output of the code later in this example.)

6. If the user has selected the Table format, use a three-phase process to output the client data. These three phases include creating a header, defining table entries for the client data, and ending the table. Once the client table is complete, the code looks at the Full argument to see if the client also wants to see the server data. If so, the code uses the same three-phase output process to create a table for the server. (We'll look at the actual output of the code later in this example.)
7. End the Web page using the EndContent( ) method. Obviously, this is the same thing we've done with all of the other examples in the chapter as well.

Designing a Web Page with ClientView( ) Access Script

We don't need a complex script to make the ClientView function work as it should, but it's important to understand that the principles you learn working with this simple example can be applied to more complex scripts. Listing 8-13 shows just the script portion of the Web page we'll use for this example, which is a standard frame setup. The script frame appears on the left side of the screen, the result appears on the right side. I'll explain the code once you've had a chance to look at it.

CD-ROM Location: Chapt08\ClientTst
Program Name: Default.HTM
Source Code Modules in Text: Select.HTM

Listing 8-13: **Script for Testing the ShowView DLL**

```
<HTML>
<HEAD>
<TITLE>Selection Page for ShowVar.DLL</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--
function MoveView()
{
     if (Table.checked)
             cFormat = 'Table'
     else
             cFormat = 'Strings'
if (False.checked)
             cFull = 'False'
     else
             cFull = 'True'
     window.parent.frames.DataView.location =
             '../controls/ShowVar.dll?ClientView?' +
             cFormat + '&' + cFull
}
//-->
</SCRIPT>
```

Continued

Listing 8-13: *(continued)*

```
</HEAD>
<BODY>

<H2>Format</H2>

<INPUT TYPE=radio
      NAME=FormatSel
      VALUE=Strings
      CHECKED ONCLICK=MoveView()
      ID=Strings>Strings<BR>

<INPUT TYPE=radio
      NAME=FormatSel
      VALUE=Table
      ONCLICK=MoveView()
      ID=Table>Table<P>

<H2>Show Server</H2>

<INPUT TYPE=radio
      NAME=FullSel
      VALUE=True
      ONCLICK=MoveView()
      ID=True>True<BR>

<INPUT TYPE=radio
      NAME=FullSel
      VALUE=False
      CHECKED ONCLICK=MoveView()
      ID=False>False<P>

</BODY>
</HTML>
```

The code for the script shown in Listing 8-13 is deceptively simple. There are four radio buttons that allow the user to choose various ISAPI Extension options (obviously, you'd have to program this element to meet the needs of your particular ISAPI Extension). All four radio buttons call the MoveView( ) function (written in JavaScript) when clicked.

The script itself is relatively simple. There are two if statements that determine which radio buttons are currently selected, and put those values into two string

variables. The next thing that happens is that we set the location of the second frame to the output of the ISAPI Extension and provide the ISAPI Extension with the user selections.

As you can see, the use of a script has allowed us to place what equals four Web pages into a single ISAPI Extension. (Theoretically, you can use additional scripts to break this example into two Web pages, but using standard techniques requires four static Web pages.) The next section of the chapter will show you the results of the ISAPI Extension and JavaScript combination.

Testing the ShowVar DLL

It's time to look at the results of all this coding. Once you create the required Web-page script (one frame page and the selection page, shown in Listing 8-13), and compile the ShowVar ISAPI Extension, you should see a result similar to the one shown in Figure 8-17. (To get the results shown in the figure, you should be using the default parameters when calling the ISAPI Extension the first time.) Notice that the output includes such things as the client's IP address and the physical location of the Web page on the server.

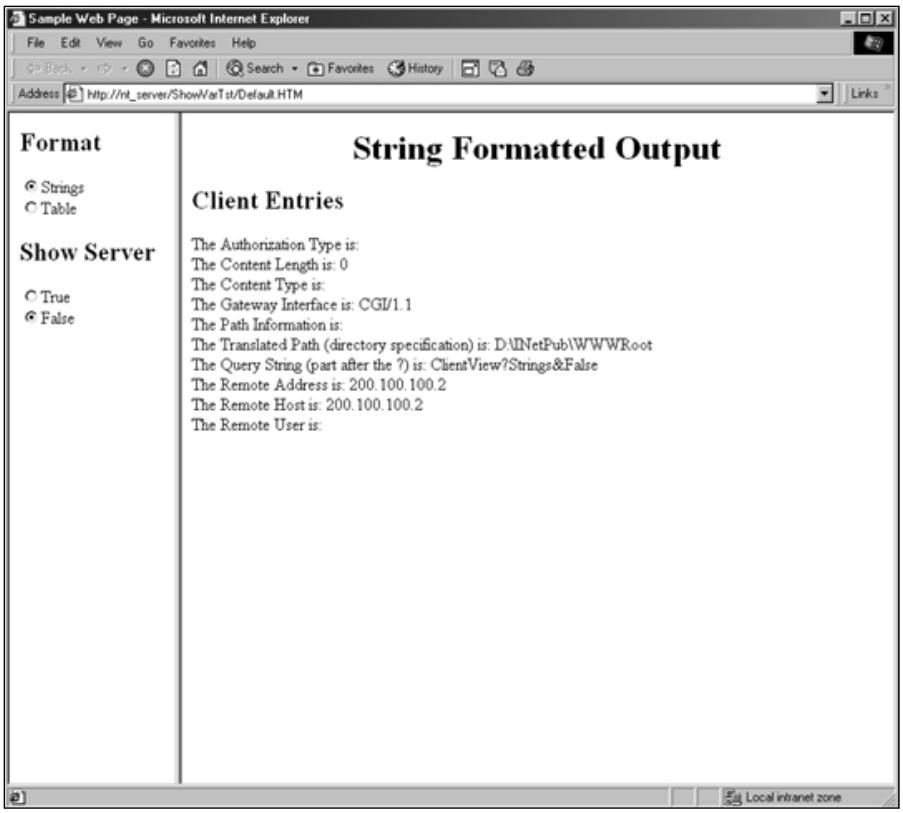


Figure 8-17: The default ShowVar ISAPI Extension output shows the Strings format without any server information.

Click the True radio button and you'll see the output shown in Figure 8-18. Notice that the server information includes the name of the ISAPI Extension being executed, the version of the HTTP that the server supports, and a list of file types that the client can accept. (This list normally doesn't include all of the file types that the client can access, it simply includes the one that the server is most likely to need at the moment.)

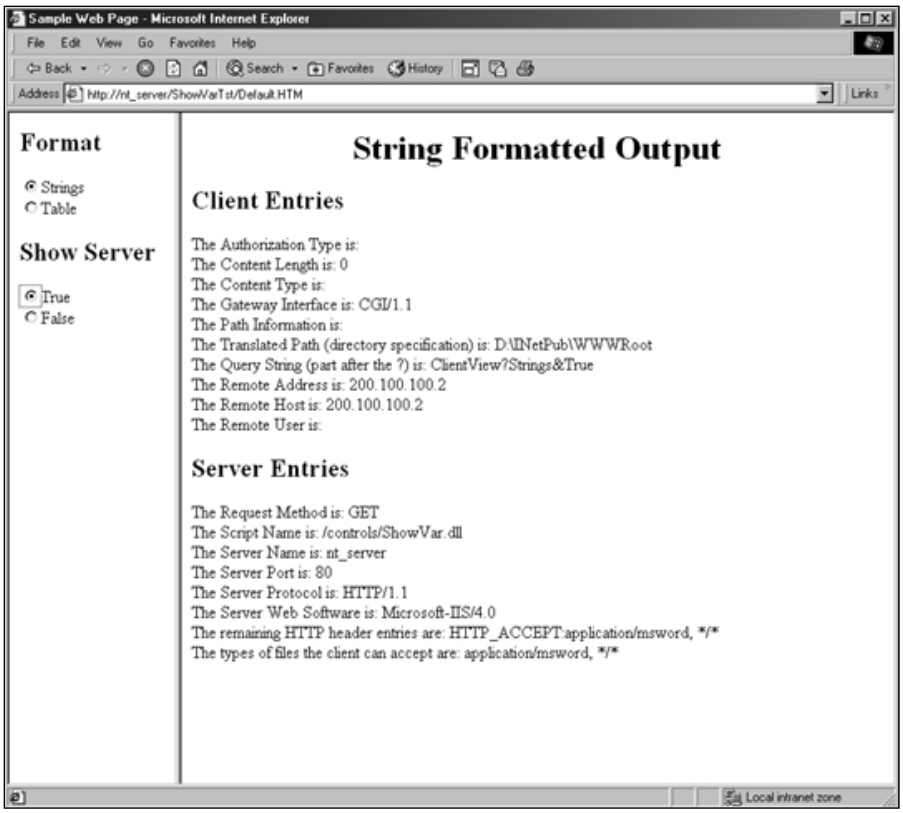

Sample Web Page - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Search Favorites History
Address http://nt_server/ShowVarTst/Default.HTM
Links
Format
Strings
Table
Show Server
True
False
String Formatted Output
Client Entries
The Authorization Type is:
The Content Length is: 0
The Content Type is:
The Gateway Interface is: CGI/1.1
The Path Information is:
The Translated Path (directory specification) is: D:\INetPub\WWWRoot
The Query String (part after the ?) is: ClientView?Strings&True
The Remote Address is: 200.100.100.2
The Remote Host is: 200.100.100.2
The Remote User is:
Server Entries
The Request Method is: GET
The Script Name is: /controls/ShowVar.dll
The Server Name is: nt_server
The Server Port is: 80
The Server Protocol is: HTTP/1.1
The Server Web Software is: Microsoft-IIS/4.0
The remaining HTTP header entries are: HTTP_ACCEPT:application/msword, */*
The types of files the client can accept are: application/msword, */*
Local intranet zone

Figure 8-18: Adding Server information to the display allows you to determine things such as the server operating system version.

If you click the Table radio button, you'll see the same information as we saw in Figure 8-18 in the rearranged format shown in Figure 8-19. It's important to remember that all of this information is always available to you with the CHttpServerContext class pointer that's passed to the function. In addition, this pointer contains unique information for every client that calls it. You should always check pCtxt to see if you're actually working with the client that you think you are. ISAPI Extensions need to provide thread safety and, as shown here, the CHttpServer Context GetServerVariable( ) method provides the means for you to get the thread-safety information that your application requires.

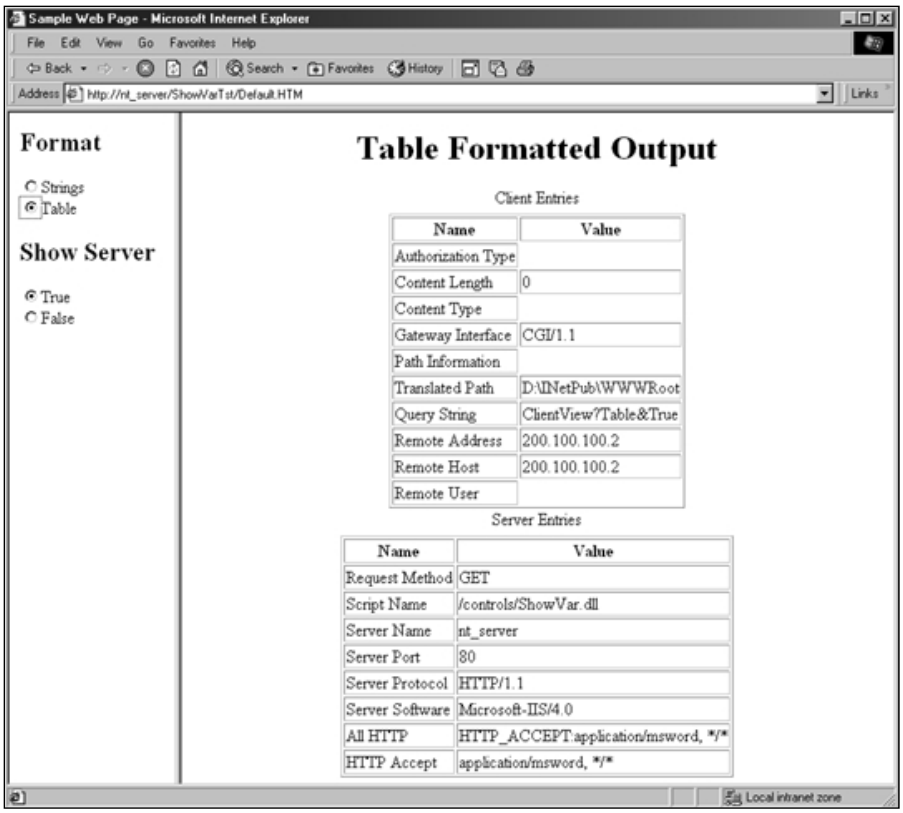


Figure 8-19: The Table format provides a different perspective on the client and server information provided by the ShowVar ISAPI Extension.

Working with Script-producing ISAPI Extensions

Just as we can use scripts to preprocess information for an ISAPI Extension, there are times when you may want to post-process the Web-page contents as well. Scripts allow you to add "fit and finish" touches to any Web page. For example, you can use scripts to respond to mouse-over events or to react in a certain way to additional user input.

Unlike the preprocessing scenario in the previous section, there aren't any advantages to producing server-side scripts with an ISAPI Extension. Any scripts

you create with an ISAPI Extension will more than likely operate at the client. In addition, they'll more than likely reside directly within the Web page.

Working with User-defined ISAPI Extensions

ISAPI Extensions provide you with the capability to use and store user settings in places other than the user's machine, where they can get lost or modified. Consider the following scenario; you want to provide secure access to your Web site. In addition, you need to provide Web content based on the user accessing the site. For example, you wouldn't want a salesperson to have access to the Web site's configuration settings, but an off-site Webmaster would require this level of access.

Most developers use cookies to store user settings for a Web site. That's fine for a lot of situations, but there are certain assumptions that such a setup makes. For one thing, the developer has to assume that the user will access the Web site from the same machine. Since the cookie is stored on the local machine, the user would have to reenter his or her settings if Web access occurs from another machine.

The other big problem with cookies is that they aren't secure. Users can modify a cookie and gain access to privileges that they don't need. For that matter, users can exchange cookies or end up with cookies that don't belong to them when using a common machine. Corrupt cookies can also cause problems by presenting information in an unreadable format, or by preventing the user from gaining the access they deserve. In essence, cookies are unreliable in the extreme and shouldn't be used for anything other than public Web sites.

We saw in a previous section of the chapter that you can add database capabilities to your ISAPI Extension. If you combine database capabilities with the ability to determine exactly which user is using the CHttpServerContext variable provided to every ISAPI Extension function, you have the combination required to ensure that the user gets the access they require. It no longer matters which machine the user tries to log in from. In addition, the security risk is minimal because all processing takes place on the server once the user provides a name and password (or a digital certificate).

There are other benefits to using a local database for security and configuration issues. The biggest benefit is to the network administrator. Access to a local database means that the administrator can set new configuration parameters or security restrictions as needed. A reconfiguration of the Web site or a change in user needs no longer means having to rework the user's hard drive or purging old cookies — everything is controlled from the server by the administrator.

Does using an ISAPI Extension with a server-based database limit the user in any way? No, the user can still upload information to the server. There are many

different methods for allowing the user to modify the contents of a database remotely, some of which can occur without the user's knowledge. In essence, using ISAPI and a local database to store user configuration settings on a private Web site gives you the best of everything.

At this point, you may be wondering why anyone would continue to use cookies. There are many good reasons to continue using cookies. For one thing, remember that ISAPI is an IIS-specific technology and the Webmaster may have a lot of UNIX servers to deal with. Second, even the best ISAPI Extension and database combination will be slower than storing a cookie on the local hard drive. Third, there's never a good reason to load the server unnecessarily if client processing is adequate. Using ISAPI and a database means adding an additional load to the server, which may not be warranted for storing things such as the user's color selections.

Summary

In this chapter, you learned how ISAPI Extensions can help you create highly customizable content. The main advantage of ISAPI Extensions over ASP scripts is speed — an ISAPI Extension is more efficient. The main disadvantage of using an ISAPI Extension is complexity — testing is difficult and writing an ISAPI Extension requires an experienced Microsoft Visual C++ programmer. Here are some additional conclusions you can draw from the contents of this chapter:

- ISAPI Extensions are DLLs that are placed on the server, called by the client, and create Web pages as output.
- The major advantages of using ISAPI Extensions and ISAPI Filters include improved execution speed, reduced code size, multiple functions per DLL, reduced memory requirements, and full access to Web-server functionality.
- The major disadvantages of using ISAPI Extensions include increased learning curve, special coding requirements, lack of debugging tools, and nonportability to other server environments.
- Modifying your current Web pages to use ISAPI Extensions can be as simple as changing the file extension used to make the call from EXE or ASP to DLL.
- ISAPI Extensions are best for performing complex tasks that require more than the usual amount of Web-server participation.
- There are five ISAPI specific functions that control the session, context, and data flow for each ISAPI Extension or ISAPI Filter request.
- The four major steps in creating an ISAPI Extension include creating the program shell, defining a parse map, declaring the client-request function, and writing the client-request function.